FTF 2016
TECHNOLOGY FORUM

DEEP DIVE INTO ODP AND DPDK FOR QORIQ LS2088A AND LS1088A

FTF-DES-N1883

HEMANT AGRAWAL
SOFTWARE ARCHITECT
FTF-DES-N1883
MAY 19, 2016

AGENDA

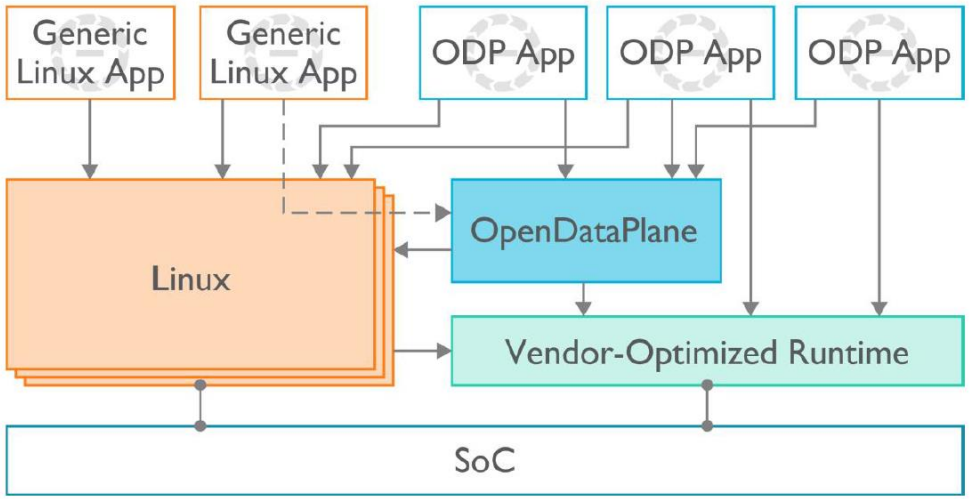
- Session Summary
- NXP Userspace Offering
- ODP
  - Overview
  - Implementation on NXP Platform
- DPDK
  - Overview
  - Support on NXP Platform
- Q &A

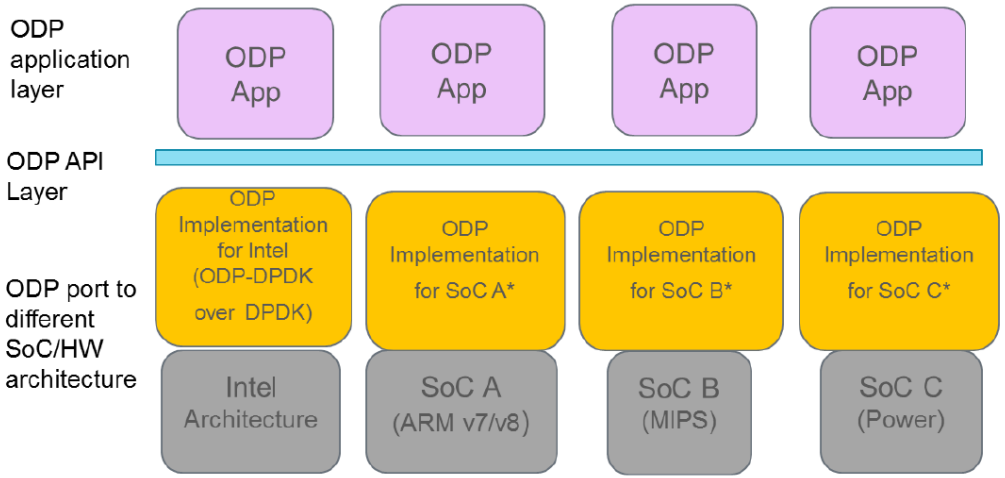
Session Objective

- SDN / NFV market drivers for a single cross-platform API standard for efficient user space packet processing
- ODP and DPDK are rapidly emerging as the open-source cross-platform API standards with fully engaged ecosystem for ARM NFV.
- NXP platform provides implementation for both while taking advantage of NXP data-path acceleration for specific packet processing elements (packet IO, classification, scheduling, crypto, etc.).
- We will also explore various possible API extensions to get additional performance on NXP SoCs.

Linaro Open Data Plane – ODP

- **Community effort**
  - Open Source, open contribution, BSD-3 licensed
  - Vendor and platform neutral (depends only on C99)
  - Application-centric--covers functional needs of data plane applications
  - Ensures portability by specifying functional behaviour of ODP
  - Defined jointly and openly by application writers and platform implementers
  - Architected to be implementable on a wide range of platforms efficiently
  - Sponsored, Governed, and Maintained by Linaro Networking Group (LNG)
- **Main focus**
  - Define a common ODP API
  - Implementations to be provided by hardware vendors (not linaro.org)
  - Allows applications to use vendor-specific extensions
- **Key features**
  - Run-time – timers, sync, memory, buffers
  - Multiple Packet I/O modes
  - Flexible queuing and scheduling
  - Crypto offload – IPSec
  - Classification and QoS

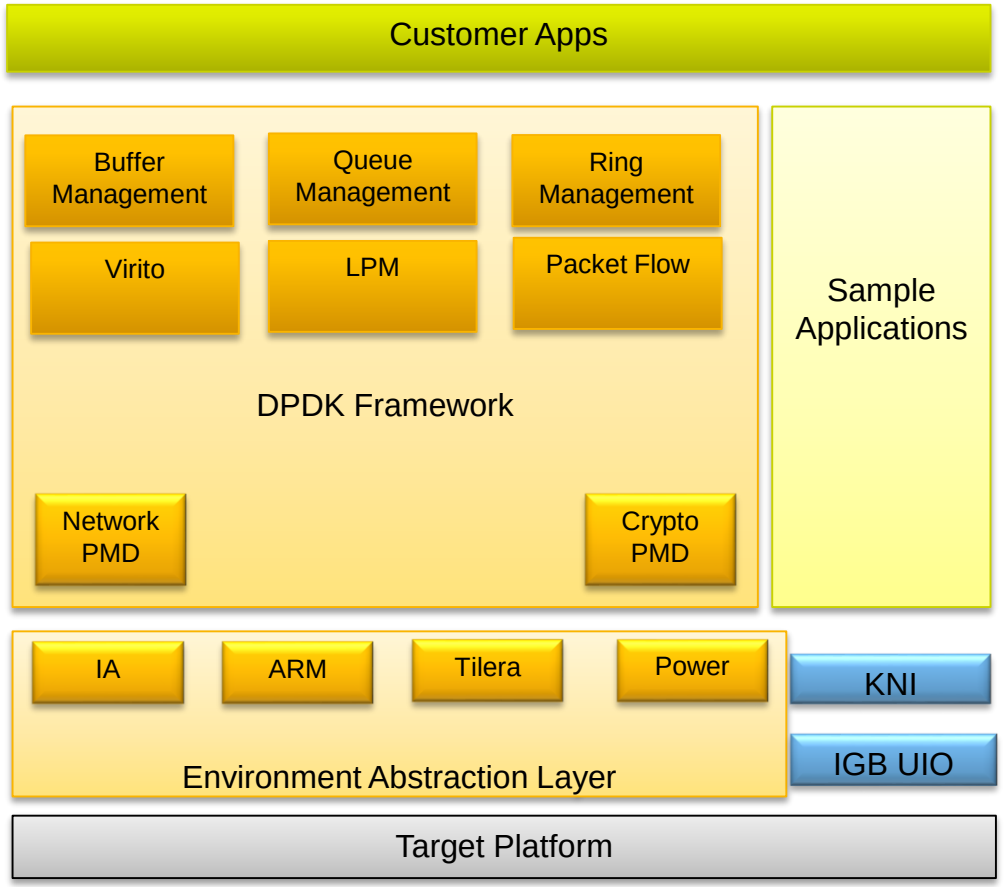




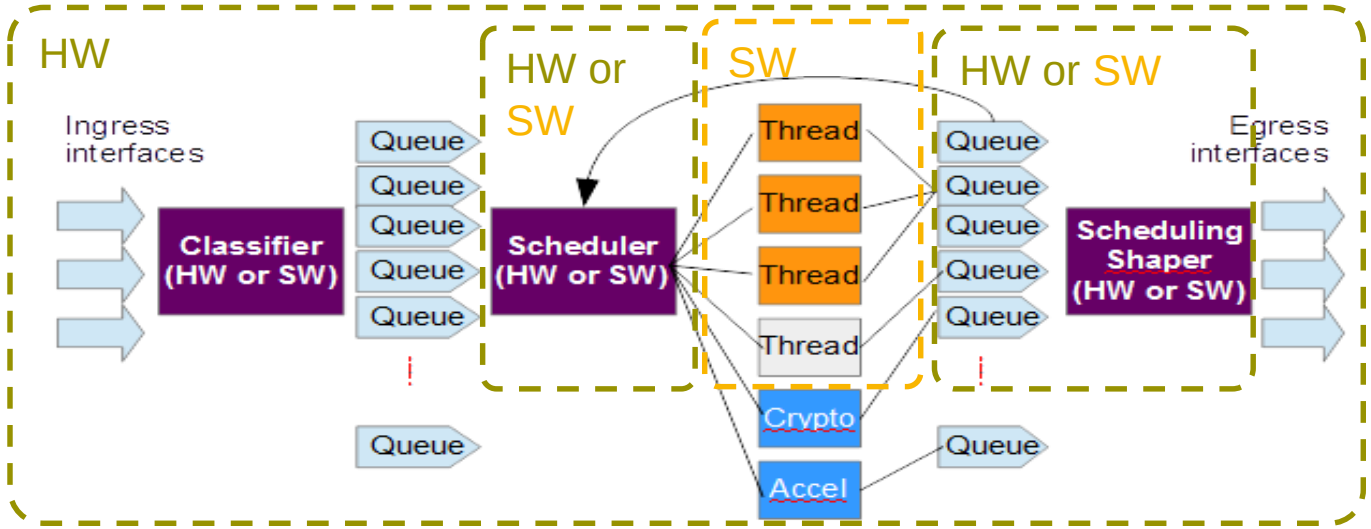
Source: "ODP Overview" by Bill Fischofer from linaro

Data Path Development Kit - DPDK

- Originally Intel initiative
  - Targeted towards SDN and NFV.
  - Large developer/user community
  - Efforts ongoing to make it part of Linux Foundation.
- Key Features
  - Core run-time libraries
    - architecture optimized run-time services
    - EAL abstracts processing model
  - Packet-access (I/O)
    - Ethernet device framework (PMD)
    - Intel NICs, virtual IO & 3rd Party NICs
  - Classification & QoS
    - Leverages HW support+ SW libraries.
  - Platform services
    - Kernel NW Interface, Power-mgmt
    - Shared-mem (inter-VM, app)
  - Crypto API
- Open to other platforms
  - Power8 as host
  - ARMv8, SoC platforms

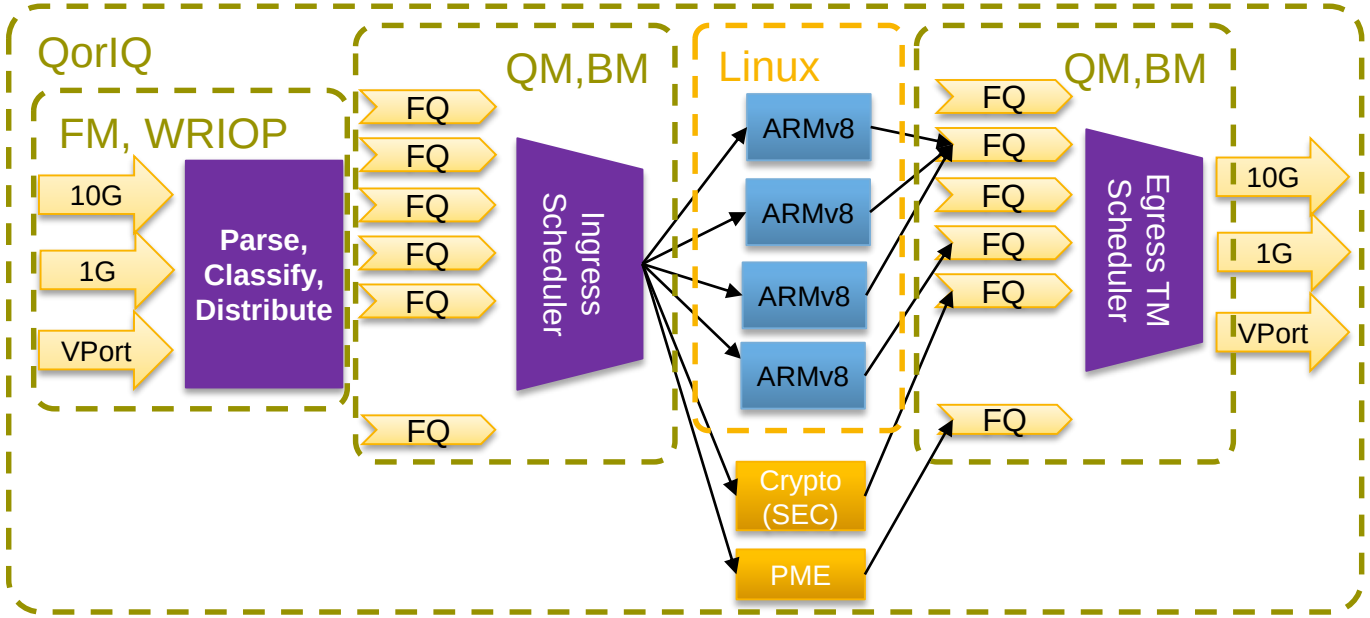


NXP DPAA – compatible with DPDK & ODP since 2008



DPDK/ODP Approach:

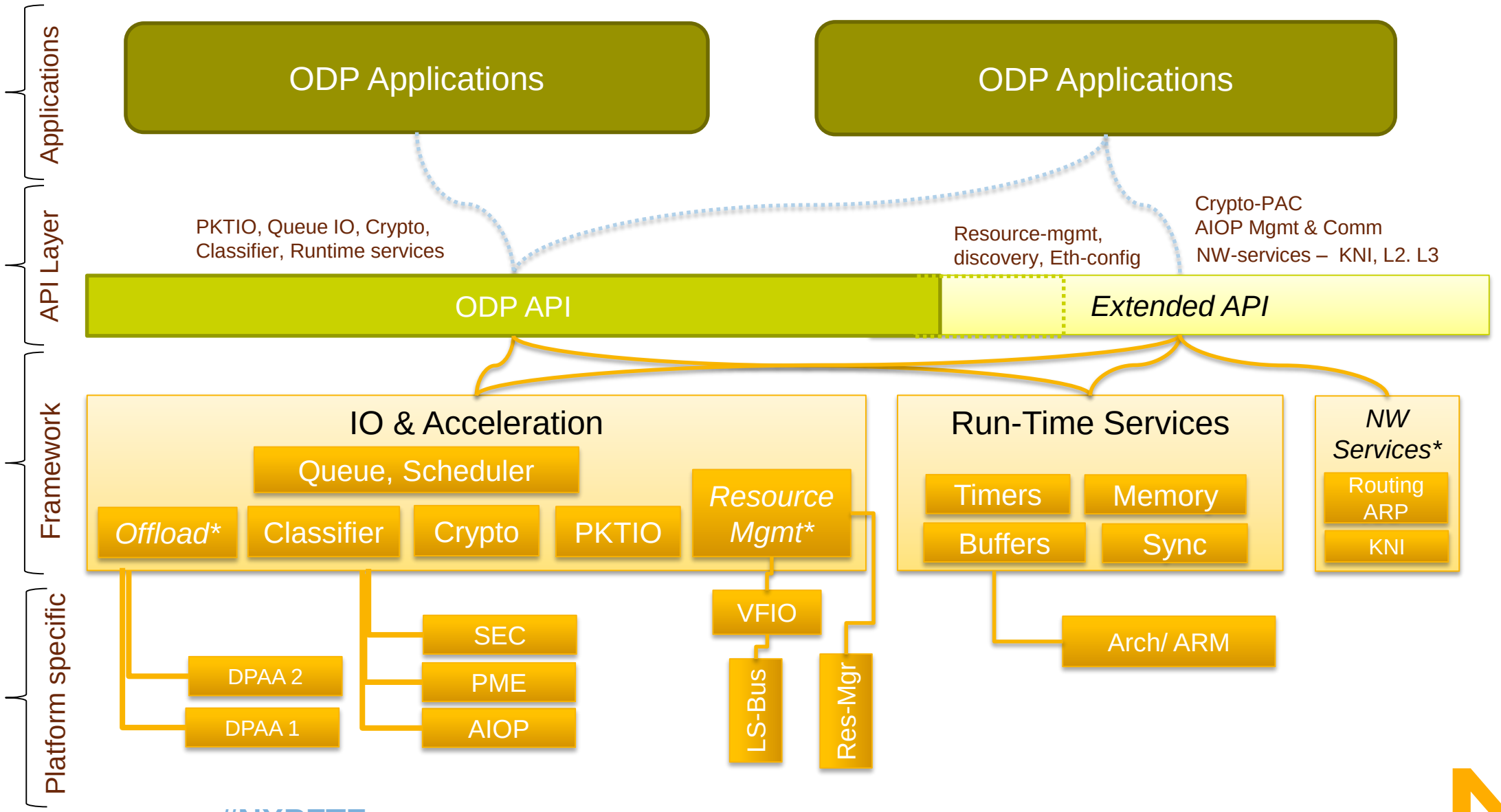
- Leverage Hardware for ingress and egress processing.
- ODP adds on HW scheduling offload
- Accelerators – Crypto offload
- Complete user-space processing model.



NXP Approach:

- FMan offloads parsing, classification, distribution.
- QMan, BMan offload scheduling, buffering
- Virtualized accelerators – SEC, PME, DCE
- User-space driver, threading model.
- Doing all this since 2008 – now into 3rd generation

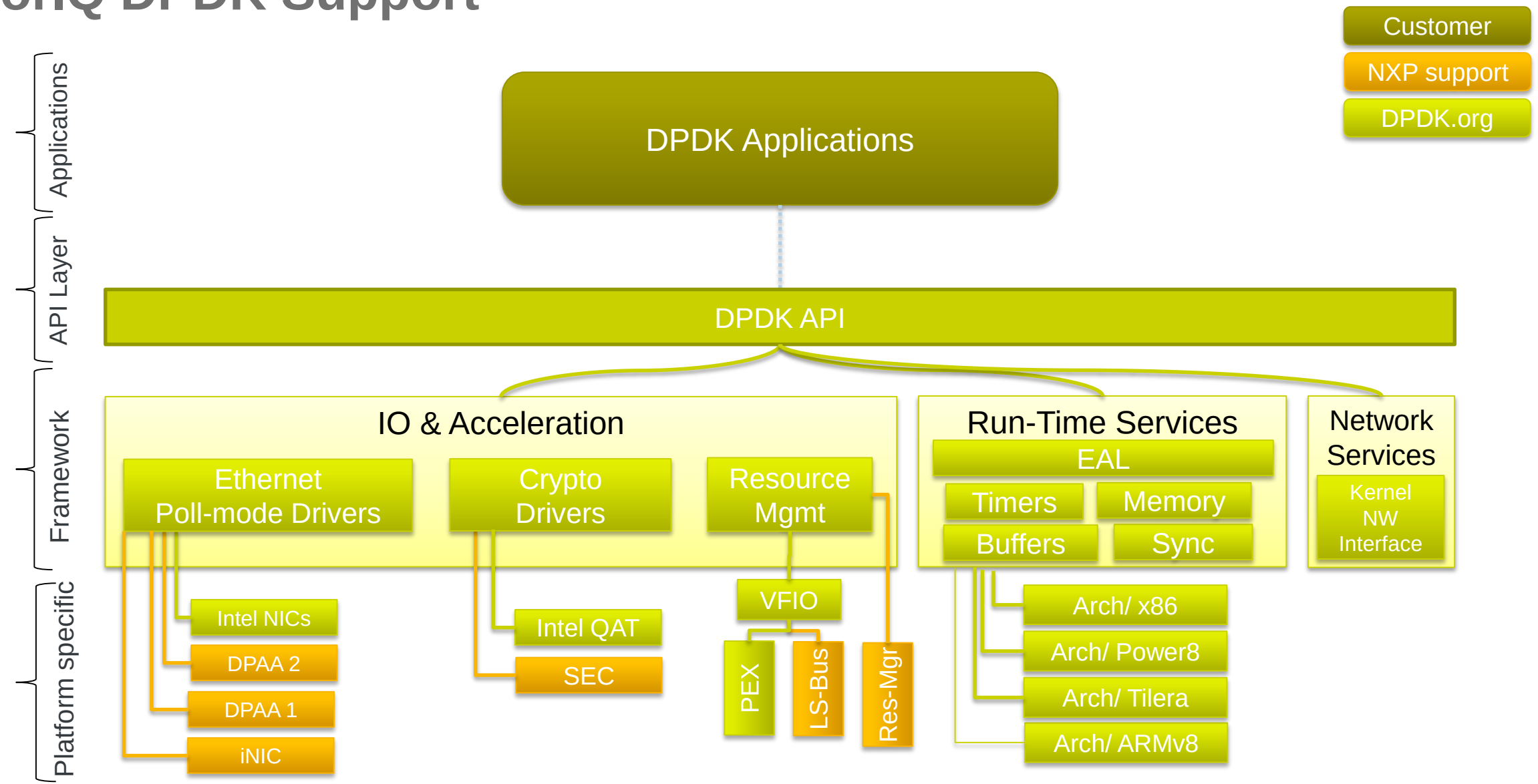
QorIQ ODP Support



QorIQ ODP Support

- QorIQ HW is inherently aligned to ODP
  - Classification and scheduling
  - HW queue and buffer mgmt
  - Crypto & other HW offloads
  - ARM 64-bit cores
- Complete ODP-API coverage
  - Queue and Scheduler API
  - PKTIO and Classifier API
  - Crypto API – algorithmic and protocol
  - Runtime services incl. pkt-buffers
  - Support for both DPAA1 & DPAA2 platforms
    - LS1043, LS1046
    - LS2088, LS1088
- QorIQ HW have additional capabilities
  - Switching, demuxing
  - Application level offloads
  - Virtual networking and resource mgmt
  - Provided as extensions to ODP-API
  - *Efforts underway to make them part of ODP*
- Value-added ODP extensions
  - Complete Ethernet capabilities
    - MAC/Phy, IPR/IPF, *GRO/GSO, Smart-NIC*
    - Physical and Virtual Ethernet ports
  - NW services
    - Provide Linux network stack services, visiblity
    - Network-devices (KNI), Routing, ARP
  - Resource management
    - VFIO and VirtIO based assignment of resources.
    - Dynamic re-configuration and discovery
    - Multiple application support, flexible process model

QorIQ DPDK Support



QorIQ DPDK Support

- Basic Platform support
  - DPAA1 Poll-mode driver
  - DPAA2 Poll-mode driver
  - iNIC Poll-mode driver
  - Crypto offload to local SEC
  - LS1043, LS1046
  - LS2080, LS2088, LS1088
  - Upstream in progress

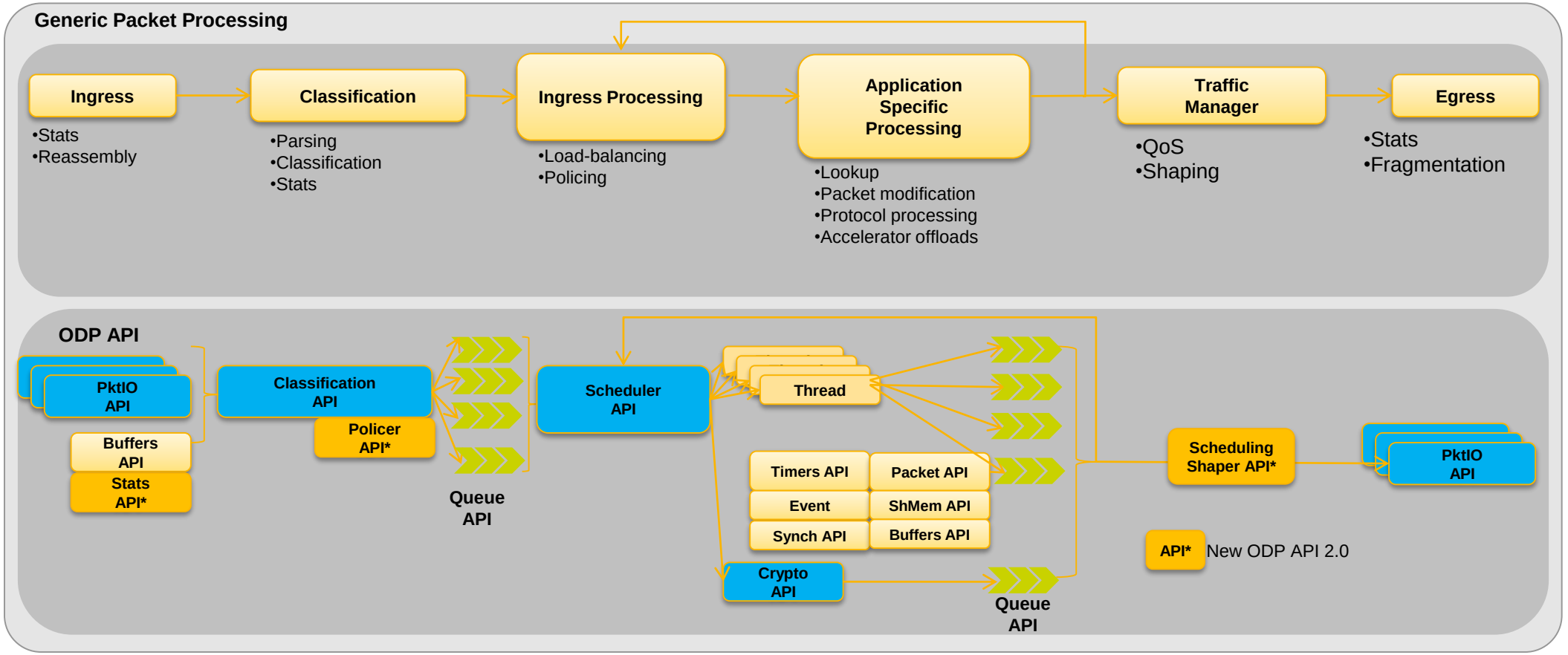
- Architectural enhancements
  - Support for SoC/platform drivers
  - Hardware buffer management
  - Optimizations for ARMv8 run-time

- Virtualization support
  - OVS over DPDK in host user-space
  - Vhost-user
  - DPDK in guest/VM
  - Virtio poll-mode driver
  - DPAA2 VFIO poll-mode driver

- Future work
  - Ingress scheduling, load balancing.
  - Protocol aware crypto
  - Egress scheduling, QoS offload.

ODP OVERVIEW

ODP API Components for Packet-processing

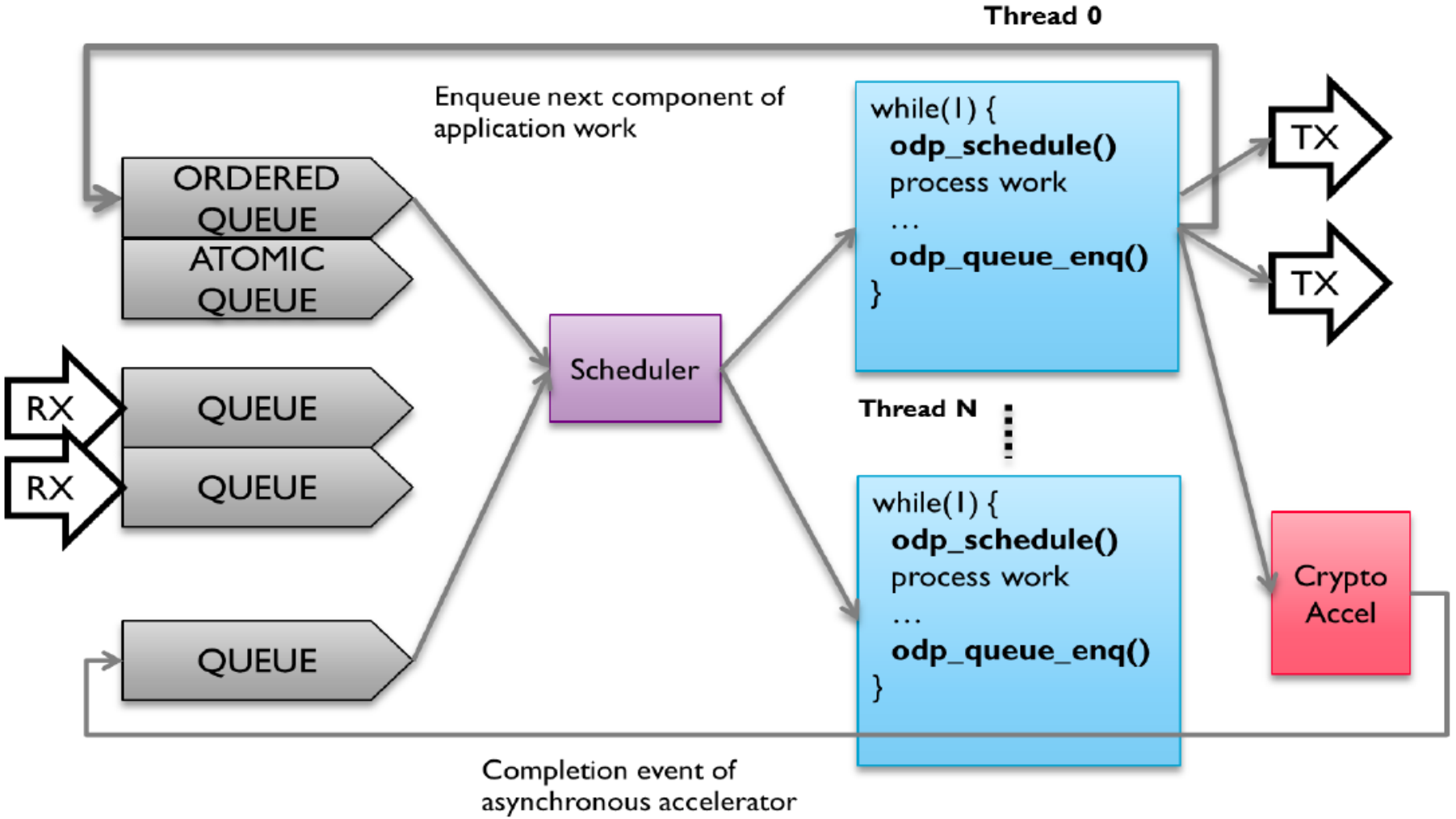


- ODP API aims to cover every aspect of packet processing pipeline
- Some components are already defined, while others are in proposal

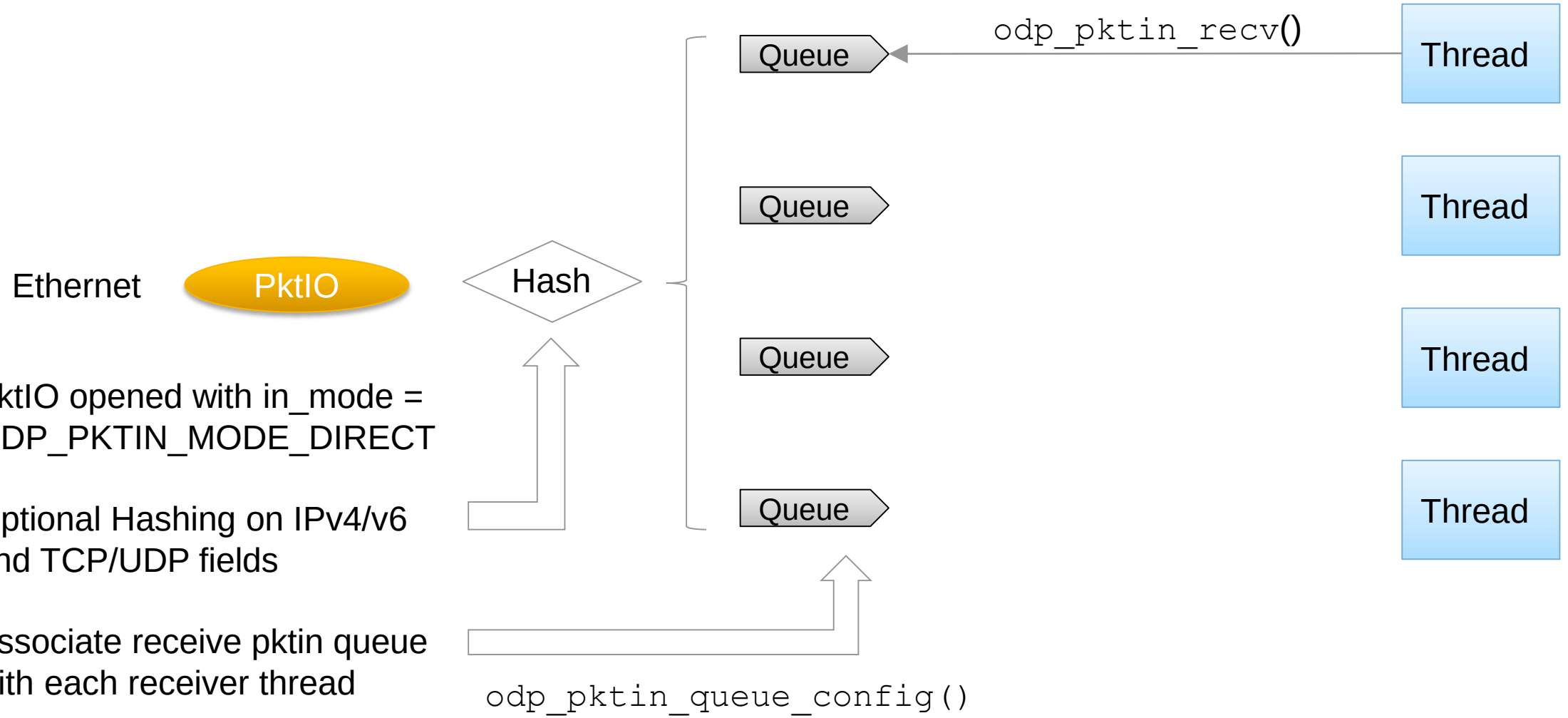
API and Concepts

- ODP loosely based on Event Machine, originally developed by NSN
  - Generic framework for scalable multi-core programming, not limited to packet processing
  - Event based abstraction and programming model for handling IO
  - Supports packet flows, physical and virtual network interfaces, accelerators, SW endpoints, etc.
  - Events represent different types of data: packets, timers, baseband data, HW notifications, SW messages...
- Supports scheduler based programming model (both HW & SW)
  - Scheduling of IO events using different algorithms and knowledge of work in progress
  - Implicit synchronisation and mutual exclusion between threads
- Supports different IO load balancing approaches
- Chose best configuration for traffic profile, latency/throughput requirements, and HW characteristics without changing the application

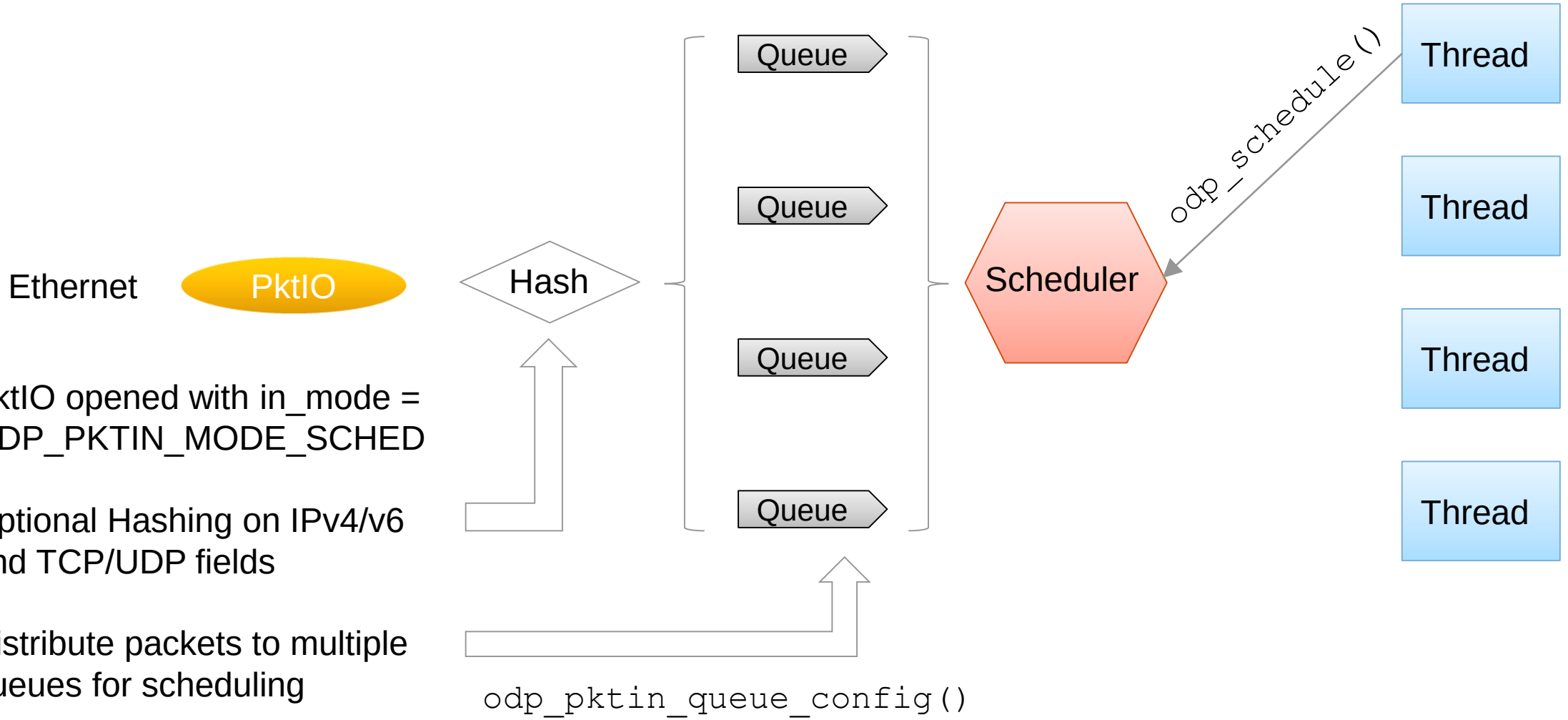
Work Scheduling and Load Distribution (Event Machine)



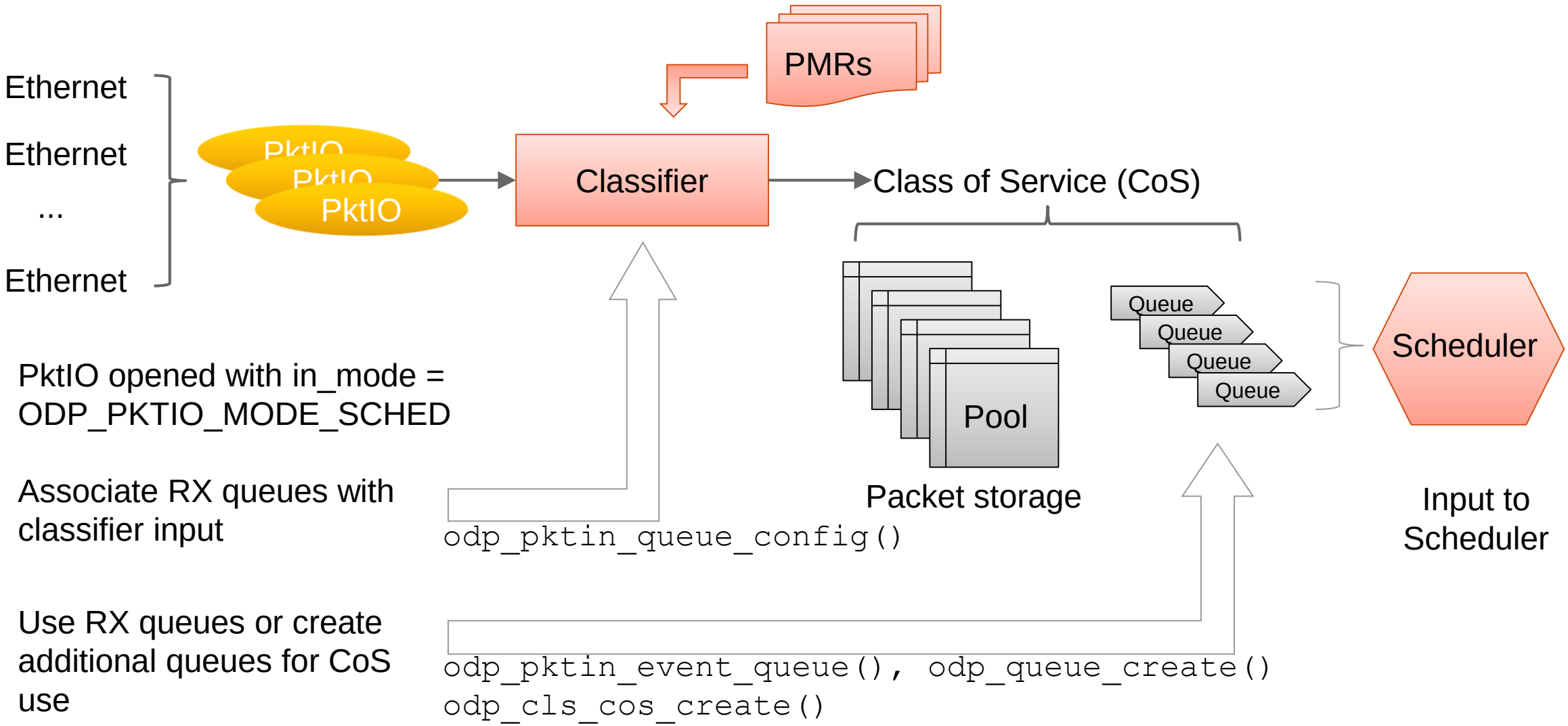
Packet Receive Options – Direct RX From Queue



Packet Receive Options – Using Scheduler



Packet Receive Options – Using Classifier – Matching Rules



PktIO opened with in_mode = ODP_PKTIO_MODE_SCHED

Associate RX queues with classifier input

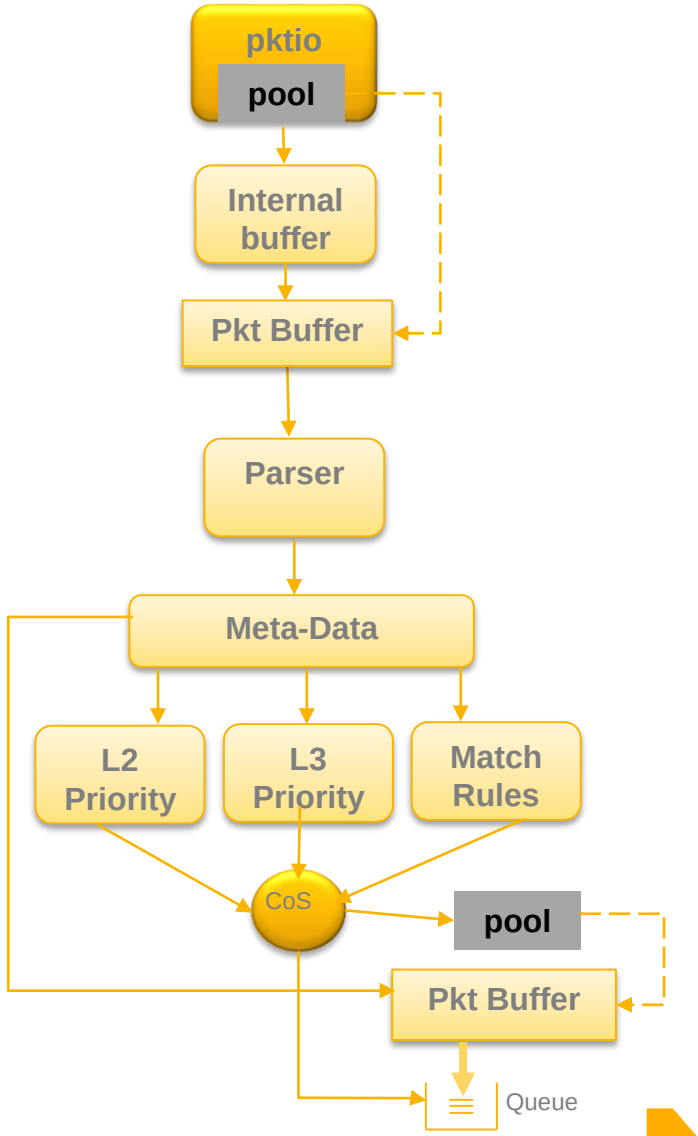
`odp_pktin_queue_config()`

Use RX queues or create additional queues for CoS use

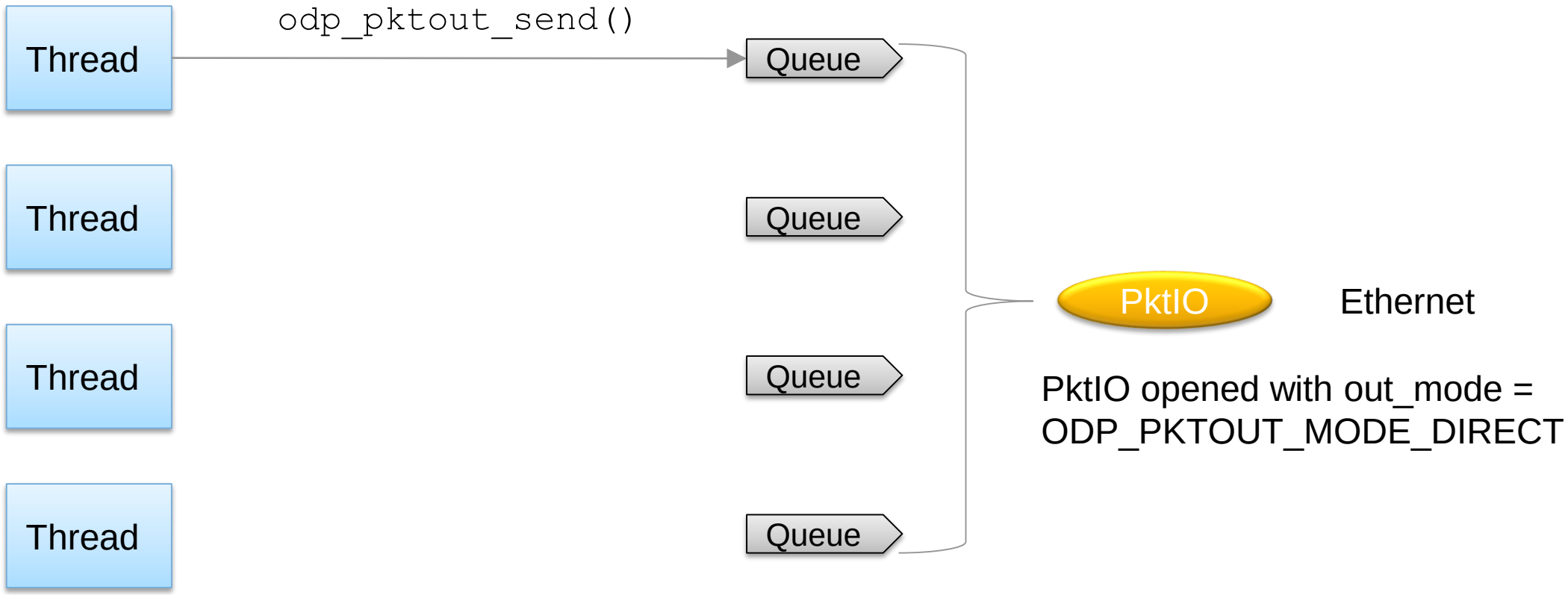
`odp_pktin_event_queue(), odp_queue_create()`
`odp_cls_cos_create()`

ODP Classification Functional Description

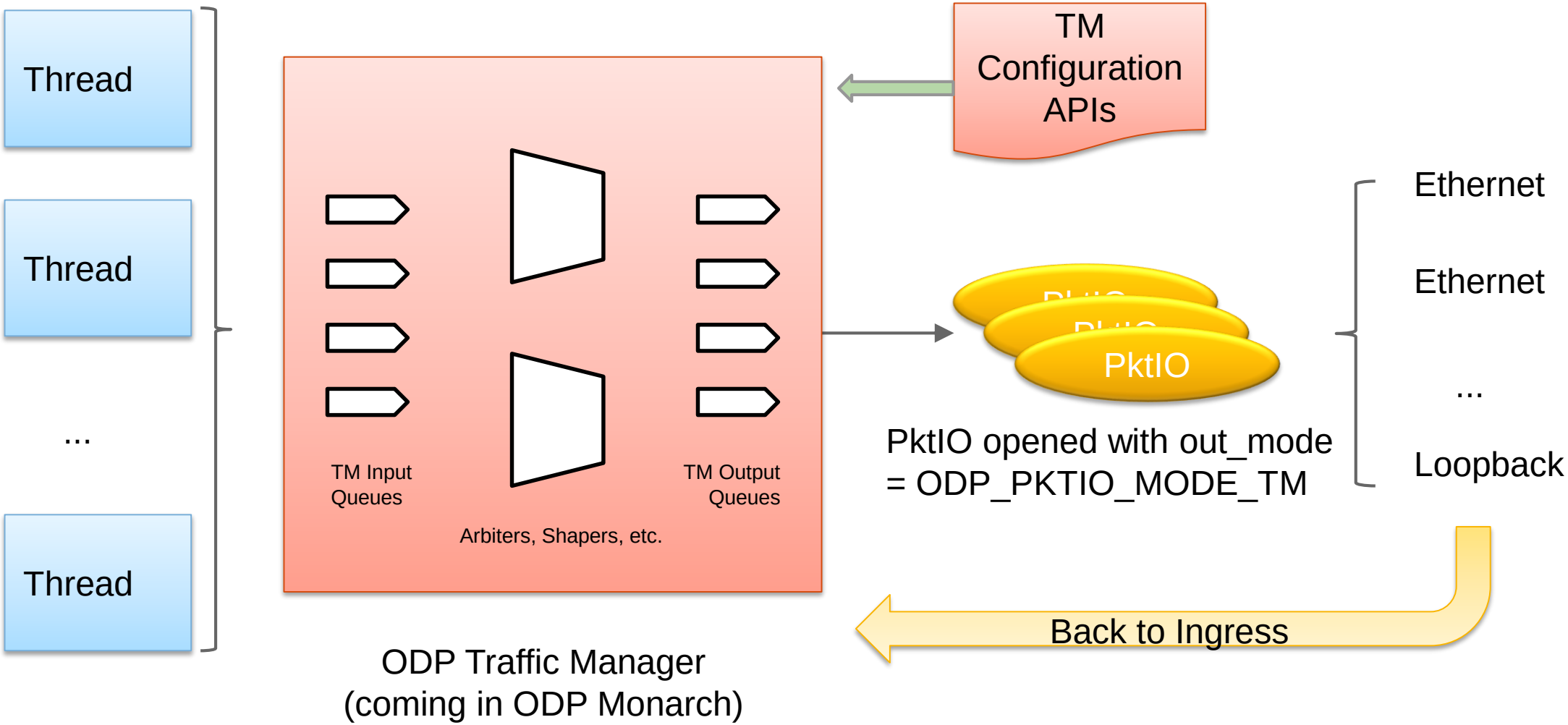
- Receive packet on pktio
- Apply set of classification rules to header of an incoming packet, identify the header fields
- Validate packet data integrity and store the validation results in packet meta-data
- Compute an odp_cos(Class of service) from 2
- Based on the odp_cos from 2), select the odp_queue
- Based on the odp_cos from 2), optionally select the odp_buffer_pool to store the packet data and meta-data
- Allocate a buffer from odp_buffer_pool selected in 6) and store the packet to the allocated buffer
- Enqueue the buffer into the odp_queue selected in 4)



Packet Transmit Options: Direct Send

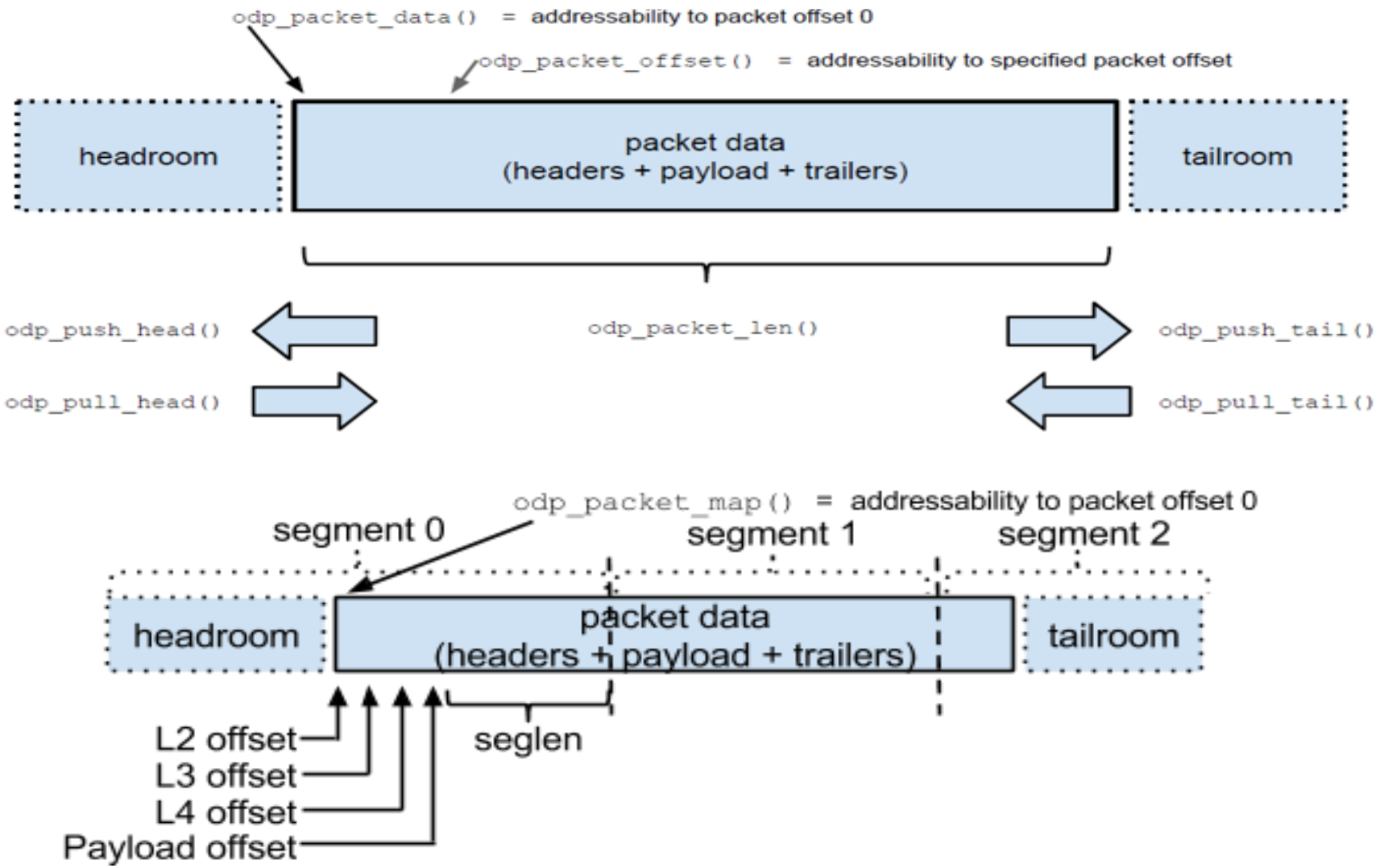


Packet Transmit Options: Egress Traffic Manager



ODP Traffic Manager
(coming in ODP Monarch)

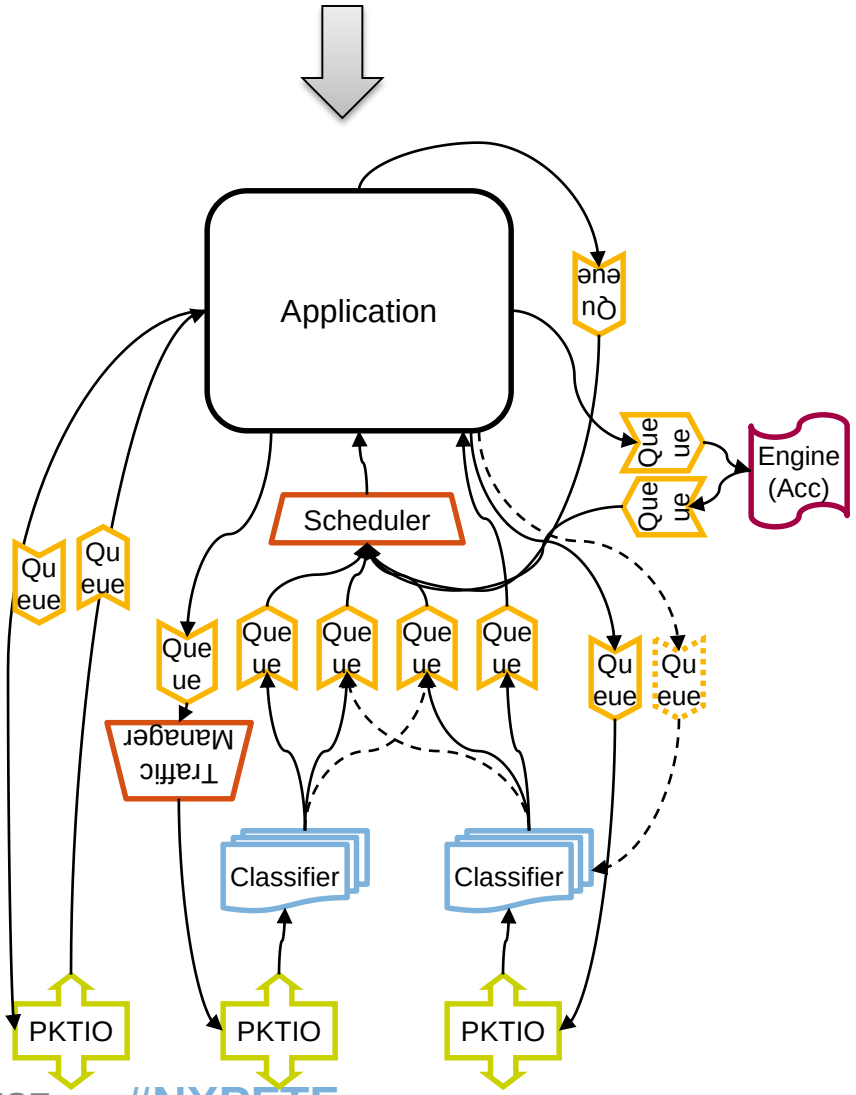
ODP Packet Buffer Format



ODP IMPLEMENTATION ON DPAA 2

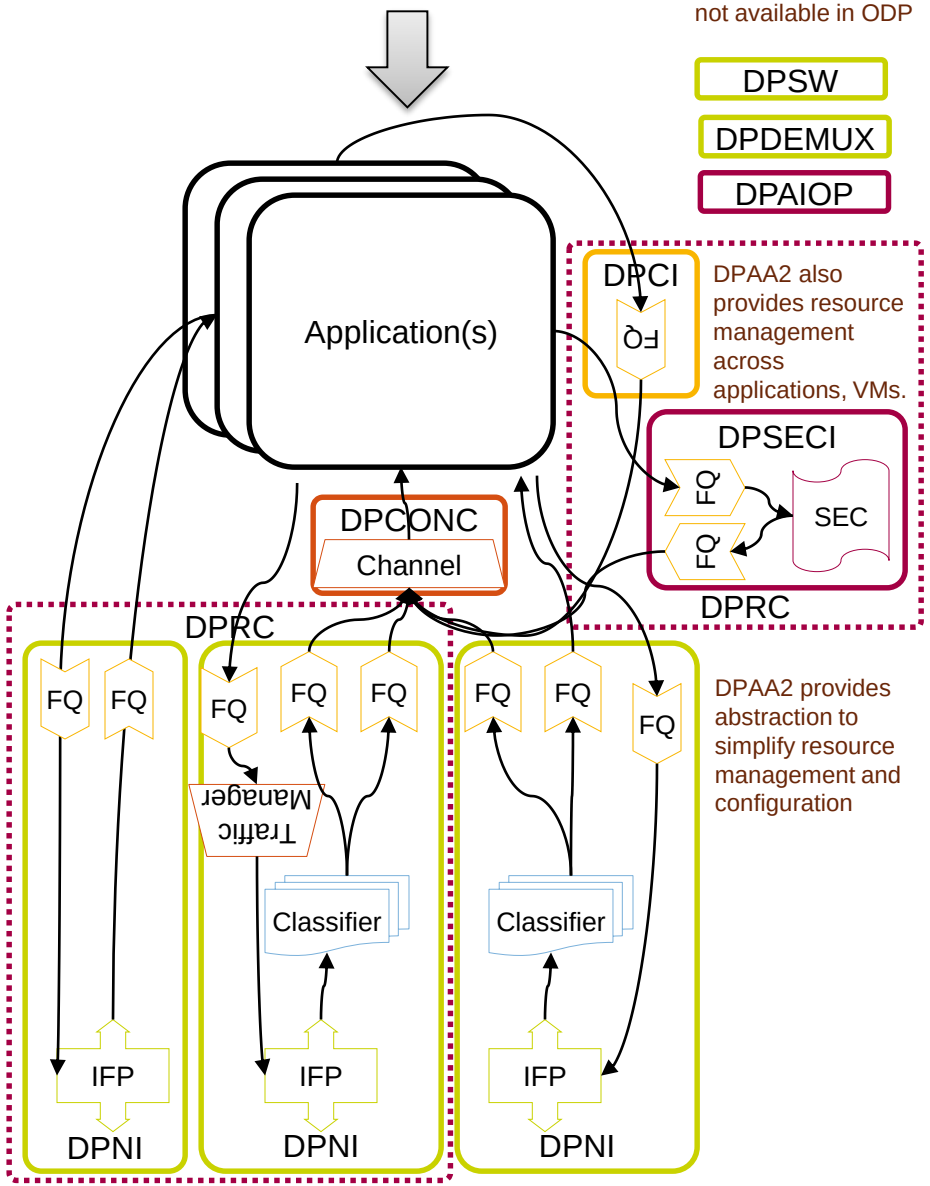
ODP APIs & its mapping to DPAA2

ODP view of the hardware



DPAA2 MC object model

DPAA2 also has other objects like DPAIOP, DPSW, DPDEMUX not available in ODP



MC Logical Objects Examples

Queues (ingress distribution width + 1 for egress) per traffic class (1 to 8)
Configuration interface
Connection point

Data Path Network Interface (DPNI) - A standard network interface (L2 and up), either basic or high-function ("smart"), as expected by standard network stacks and other network applications.

QBMan channel for dequeue
Configuration interface
Priorities for scheduling ingress queues (2-8)

Data Path Concentrator (DPCON) – Scheduling object for advanced scheduling of ingress packets from multiple interfaces.

GPP interface (QBMan software portal)
Configuration interface
Notification channel (optional: notifications from ingress queues or DPCON)

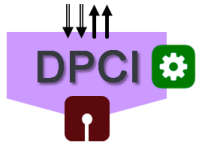
Data Path I/O (DPIO) - For performing enqueue/dequeue via QMan portals and receiving data available notifications

Configuration interface

Data Path Buffer Pool (DPBP) – An abstraction of BMan buffer pool

Queues (1 for ingress + 1 for egress) per priority (1 to 8)
Configuration interface

SEC Accelerator Interface (DPSECI)

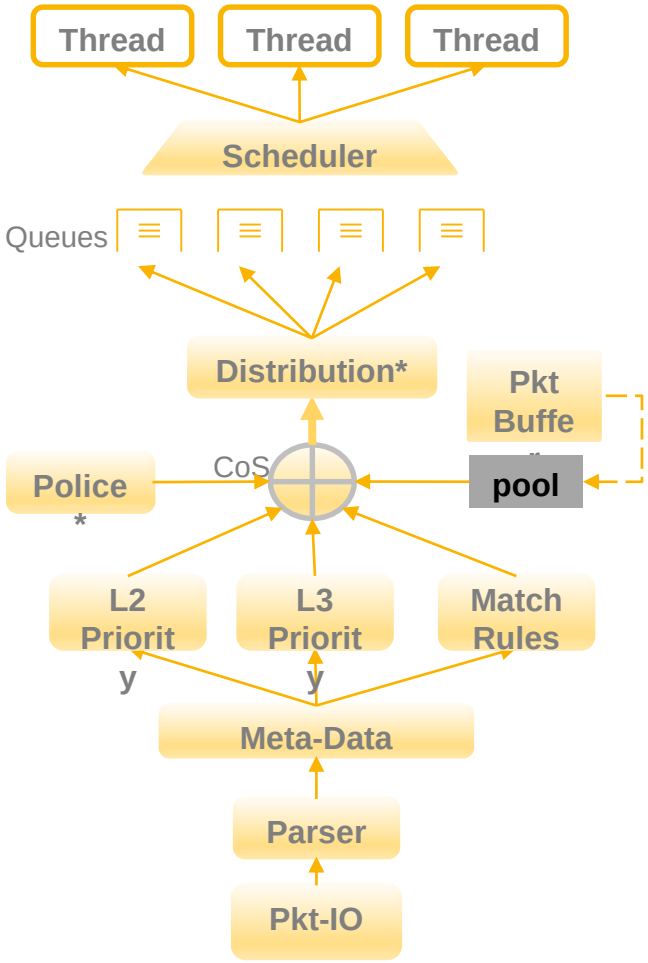


Queues (1 for ingress + 1 for egress) per priority (1 or 2)
Configuration interface
Connection point (to peer DPCI only)

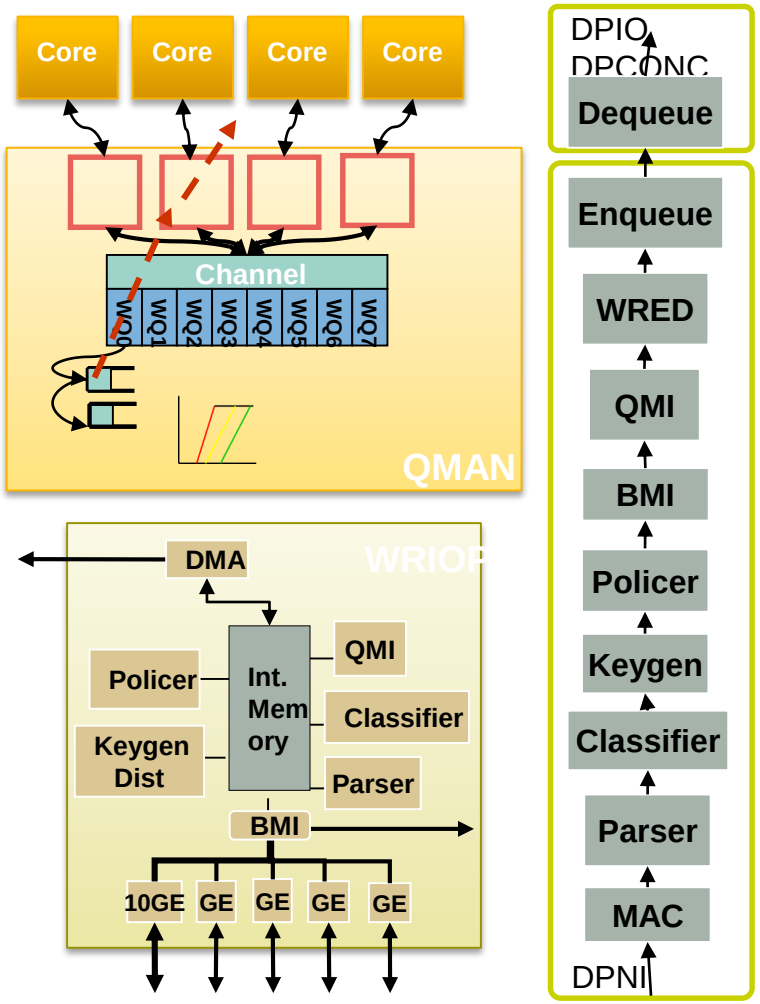
Data Path Communication Interface (DPCI) – allows communication between different software contexts through QMan infrastructure, which is not limited to network packet format. Useful for IPC between two GPP software entities, or between GPP and AIOP entities. The communication protocol is user-defined.

Mapping ODP Ingress Processing to DPAA-2

ODP Ingress processing

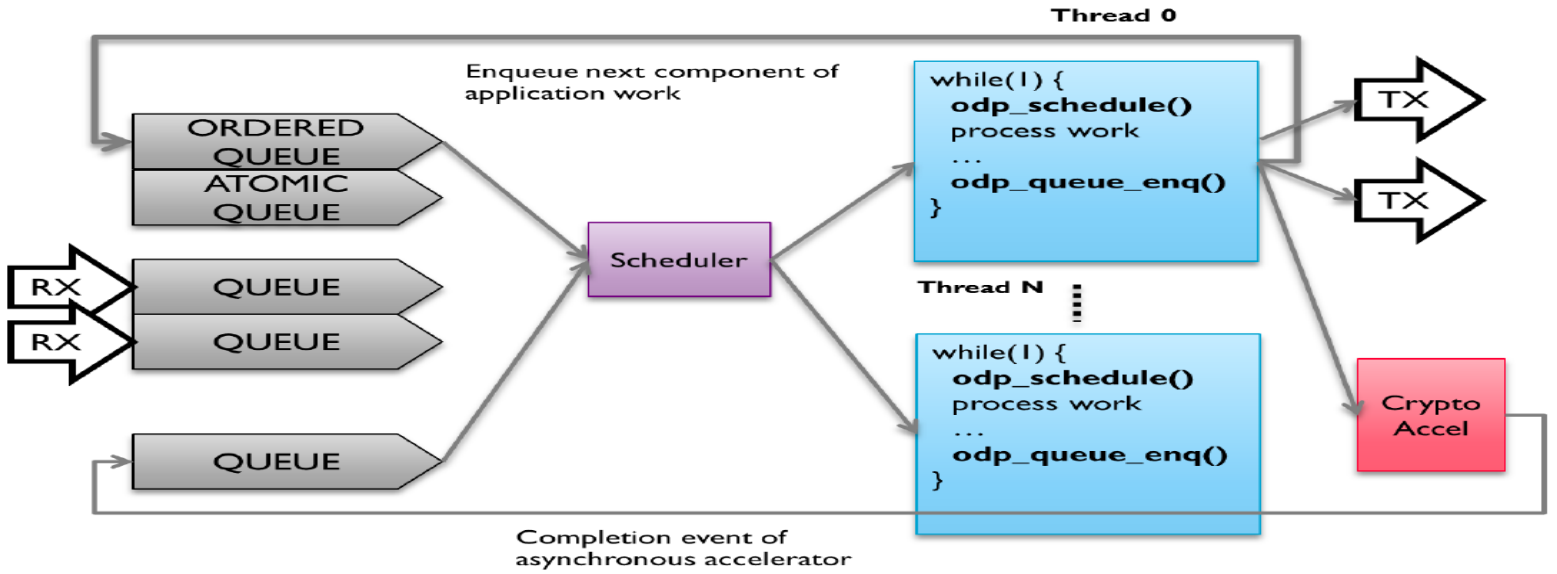


DPAA 2 Ingress processing

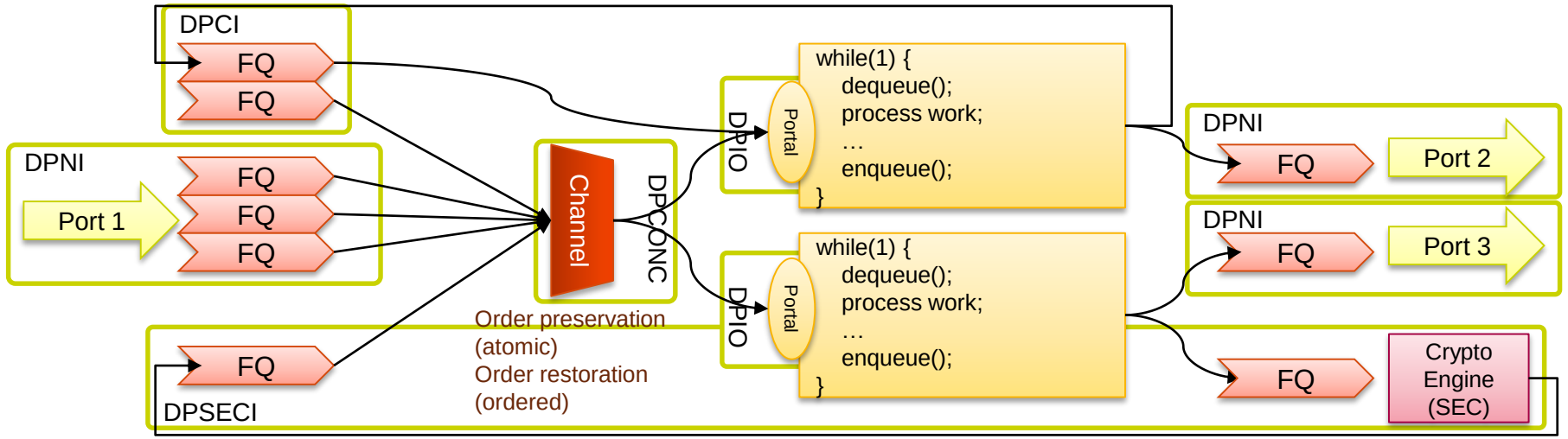


Mapping ODP Centralized Queuing/Scheduling to DPAA-2

ODP queuing & scheduling



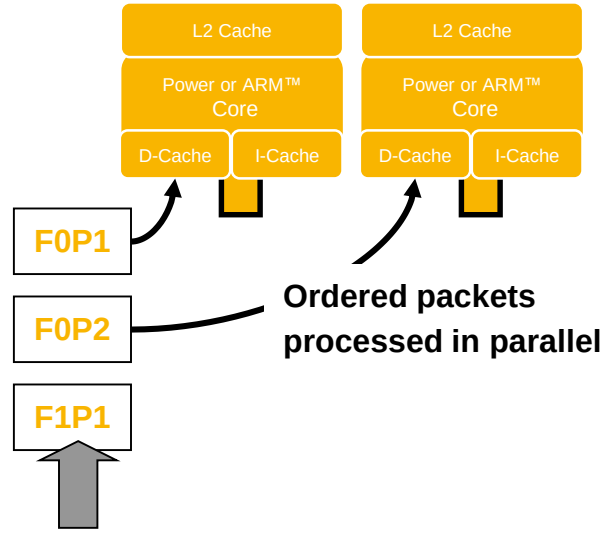
DPAA queuing & scheduling

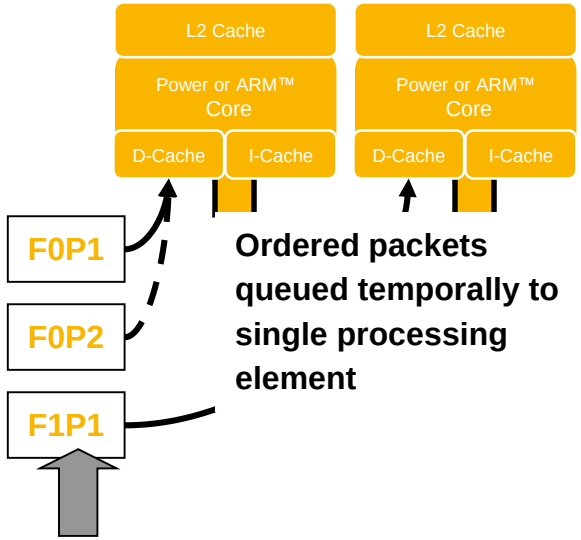


Mapping ODP HW Flow ordering and Synchronization

- There are two basic approaches:
  - Order restoration (a.k.a ODP ‘ordered’ queues)
    - Take note of the correct order (or sequence) of packets before processing starts and restore the packets to that order before they are transmitted
  - Order preservation (a.k.a ODP ‘atomic’ queues)
    - Ensure that related packets are processed in order (and typically one at a time)
    - Order preservation can also provide “atomicity” – atomic access to data used in processing the frame

- QMan requires that related frames (which must be transmitted in order) be placed on the same frame queue for both of these approaches
  - This does not mean that only related frames are placed on a given FQ
  - Many sets of related frames can be placed on an FQ
  - WRIOP is responsible for achieving this

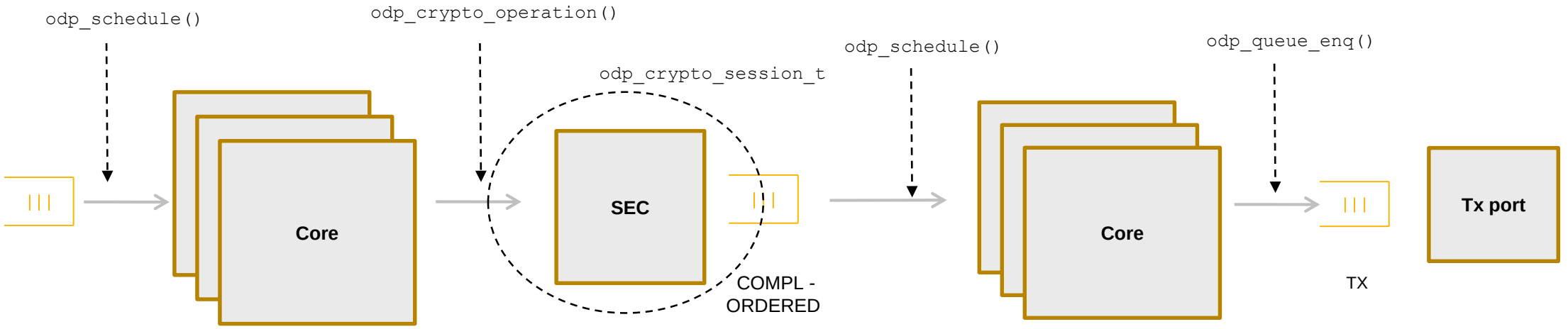




Cryptographic Services

- Provide ciphering, authentication, random number generation
- Basic concepts: sessions & operations
- Crypto sessions
  - All packets/buffers processed by a session share the parameters that define the session
    - int odp_crypto_session_create(odp_crypto_session_params_t *params, odp_crypto_session_t *session, enum odp_crypto_ses_create_err *status)
  - Session parameters: algorithms, keys, IVs, encode or decode, sync or async, output queue for async, etc
- Crypto operations
  - Performs the operations specified during session creation
    - int odp_crypto_operation(struct odp_crypto_op_params *params, odp_bool_t *posted, odp_crypto_op_result_t *result ODP_UNUSED)
  - Operation parameters: session, input/output packets/buffers, ranges in packets, operation context
  - Async operation sets *posted
  - A completion event is delivered from the session completion queue
  - Completion event provides the result of the operation — status, output packet, context

ODP Crypto Processing…



IPSEC Protocol Offload APIs

- Configure existing crypto session for IPSEC protocol offload
- Configures a crypto session for IPSec protocol processing.
- add/remove of ipsec headers/trailers and tunnel header.
- Internal maintenance of esp sequence numbers.
- Optional parameters, based on underlying hardware capabilities.

```
typedef struct odp_ipsec_params {
    uint32_t spi;        /** SPI value */
    uint32_t seq;        /** Initial SEQ number */
    enum odp_ipsec_ar_ws ar_ws; /** Anti-replay window size -
                   inbound session with authentication */
    odp_bool_t esn;      /** Use extended sequence numbers */
    odp_bool_t auto_iv; /** Auto IV generation for each operation. */
    uint16_t out_hdr_size;   /** outer header size - tunnel mode */
    uint8_t *out_hdr;    /** outer header - tunnel mode */
    enum odp_ipsec_outhdr_type out_hdr_type; /* outer header type -
                         tunnel mode */
    odp_bool_t ip_csum; /** update/verify ip header checksum */
    odp_bool_t ip_dttl; /** decrement ttl - tunnel mode encap & decap */
    odp_bool_t remove_outer_hdr; /** remove outer header - tunnel mode decap */
    odp_bool_t copy_dscp;   /** DiffServ Copy - Copy the IPv4 TOS or
                   IPv6 Traffic Class byte from the inner/outer
                   IP header to the outer/inner IP header -
                   tunnel mode encap & decap */
    odp_bool_t copy_df; /** Copy DF bit - copy the DF bit from
                   the inner IP header to the
                   outer IP header - tunnel mode encap */
    odp_bool_t nat_t;   /** NAT-T encapsulation enabled - tunnel mode */
    odp_bool_t udp_csum;    /** Update/verify UDP csum when NAT-T enabled */

} « end odp_ipsec_params »  odp_ipsec_params_t;
```

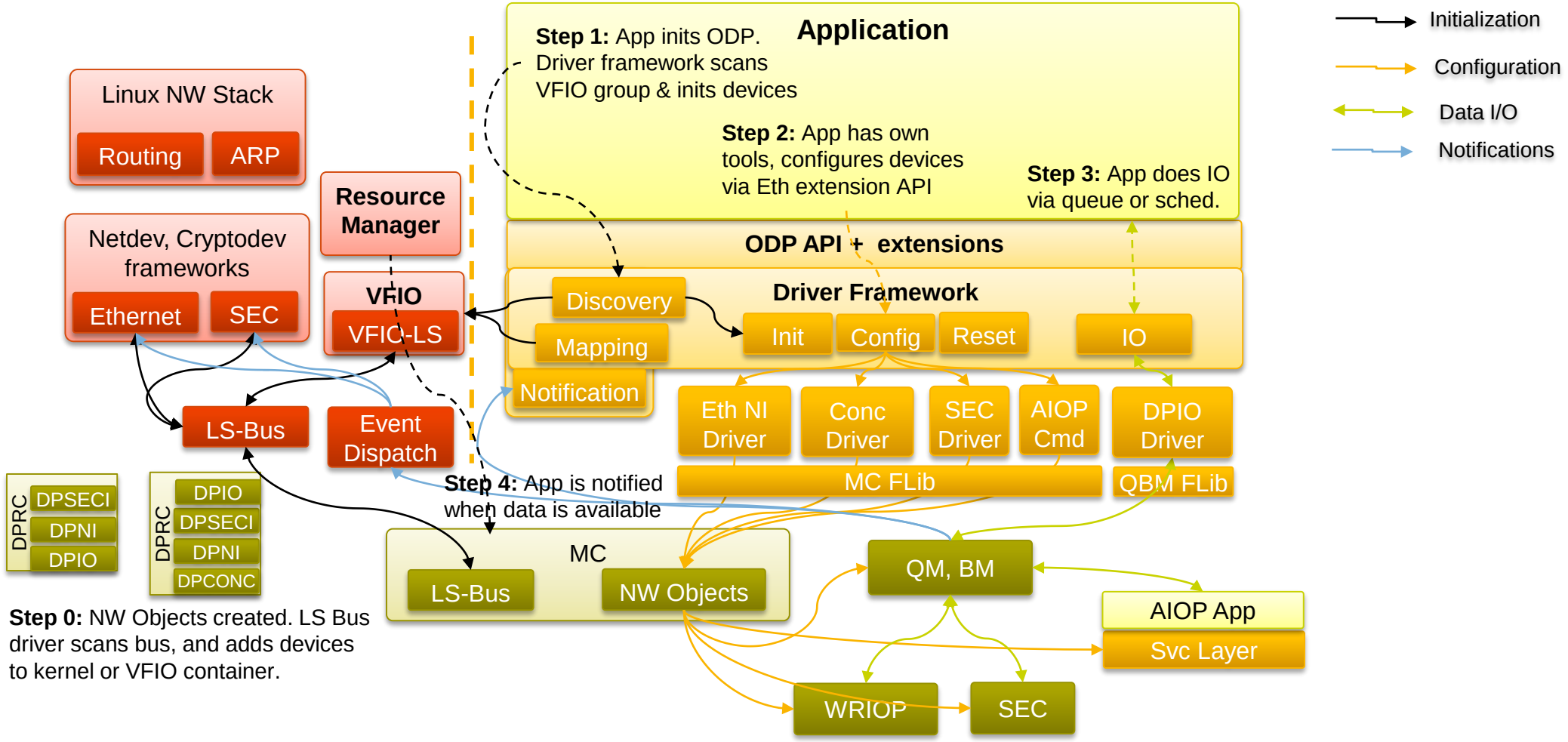
```
 * @param session       Session handle
 * @param ipsec_mode        IPSec protocol mode
 * @param ipsec_proto       IPSec protocol
 * @param ipsec_params      IPSec parameters. Parameters which are not
 *              relevant for selected protocol & mode are ignored -
 *              e.g. outer_hdr/size set for ESP transport mode.
 * @retval 0 on success
 * @retval <0 on failure
 */
int odp_crypto_session_config_ipsec(
                    odp_crypto_session_t session,
                    enum odp_ipsec_mode ipsec_mode,
                    enum odp_ipsec_proto ipsec_proto,
                    odp_ipsec_params_t *ipsec_params);
```

PDCP Protocol Offload APIs

- Configure existing crypto session for PDCP protocol offload
- - supports both control and data mode
- The IV needed for auth/cipher will be internally generated.
- Add ICV trailer/compare ICV trailer
- HFN maintenance and HFN threshold notification
- HFN over-ride per operation

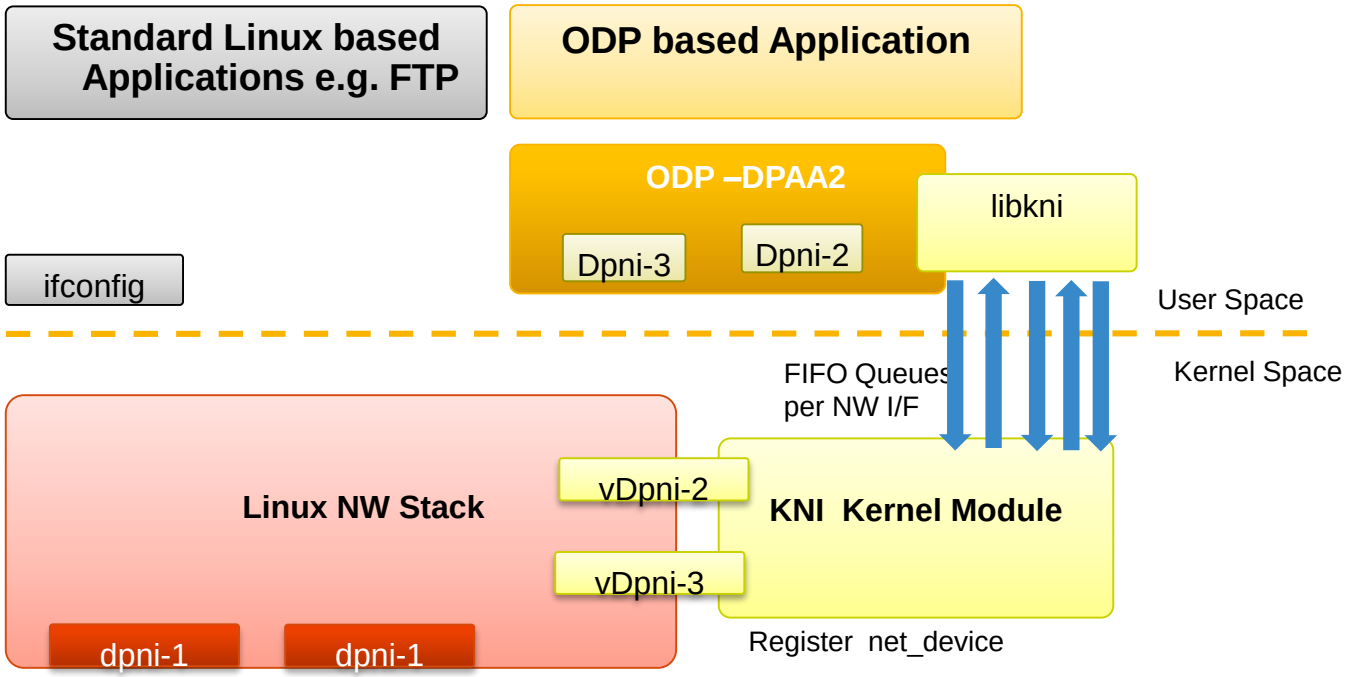
```
/** @addtogroup odp_crypto
 *  @{
 */
enum odp_pdcp_mode {
    ODP_PDCP_MODE_CONTROL,      /**< PDCP control plane mode */
    ODP_PDCP_MODE_DATA,   /**< PDCP data plane mode */
};

/**
 * The structure is to be filled by user as a part of
 * odp_sec_proto_ctxt for PDCP Control or Data Plane Protocol
 */
typedef struct odp_pdcp_params_s {
    int8_t bearer;  /**< PDCP bearer ID */
    int8_t pkt_dir; /**< PDCP Frame Direction 0:UL 1:DL*/
    int8_t hfn_ovd; /**< Overwrite HFN per operation*/
    uint8_t sn_size; /**< Sequence number size, 5/7/12/15 */
    uint32_t hfn;   /**< Hyper Frame Number */
    uint32_t hfn_threshold; /**< HFN Threashold for key renegotiation */
} odp_pdcp_params_t;

/**
 * Configures a crypto session for PDCP protocol processing.
 *
 * If an implementation does not support a particular set of
 * arguments it should return error.
 *
 * @param session       Session handle
 * @param pdcp_mode     Control Plane or Data Plane
 * @param pdcp_params       PDCP parameters.
 * @retval 0 on success
 * @retval <0 on failure
 */
int odp_crypto_session_config_pdcp(odp_crypto_session_t session,
                  enum odp_pdcp_mode pdcp_mode,
                   odp_pdcp_params_t* pdcp_params);

/**
 * Crypto API operation result
 */
typedef struct odp_crypto_op_result {
    odp_bool_t  ok;                 /**< Request completed successfully */
    void *ctx;                      /**< User context from request */
    odp_packet_t pkt;               /**< Output packet */
    odp_crypto_compl_status_t cipher_status; /**< Cipher status */
    odp_crypto_compl_status_t auth_status;   /**< Authentication status */
    int proto_status;       /**< Protocol specific status*/
} odp_crypto_op_result_t;
```

ODP Extensions – Device Mgmt

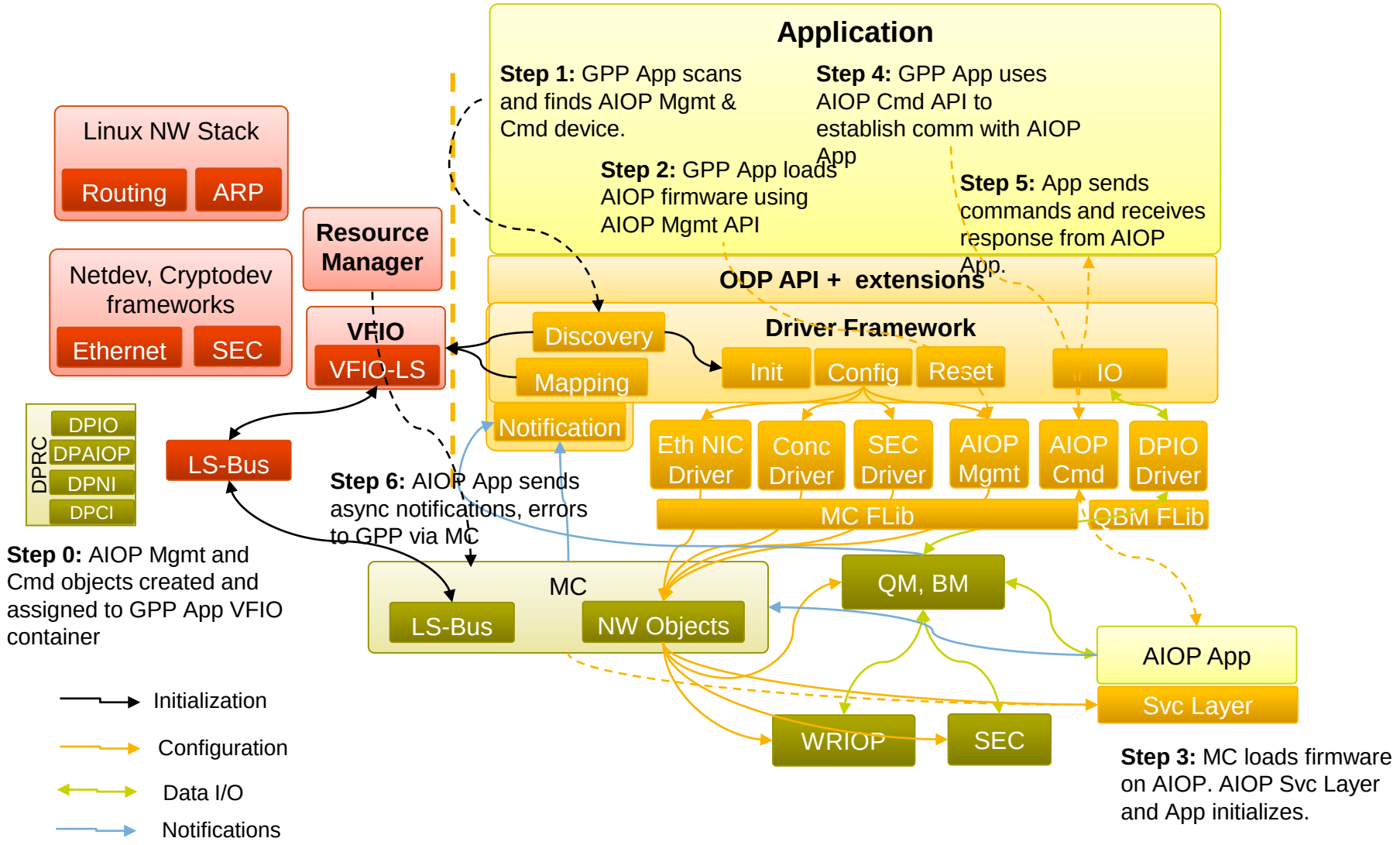


Kernel Network Interface

- The Kernel NIC Interface (KNI) is an ODP extn control plane solution that allows user space applications to exchange packets with the kernel networking stack. It creates a pseudo network interface in kernel for any network interface in ODP(e.g. WRIOP- DPNI).
  - The ODP extn creates a FIFO queue for packet ingress and egress to the KNI kernel module for each device created.
  - Kernel Network Interface provides a mechanism between kernel and user-space components to send/receive control/management packets to the Linux network stack. (copy-based)
  - It is destroyed on application exit.



ODP Extensions – AIOP Mgmt & Communication

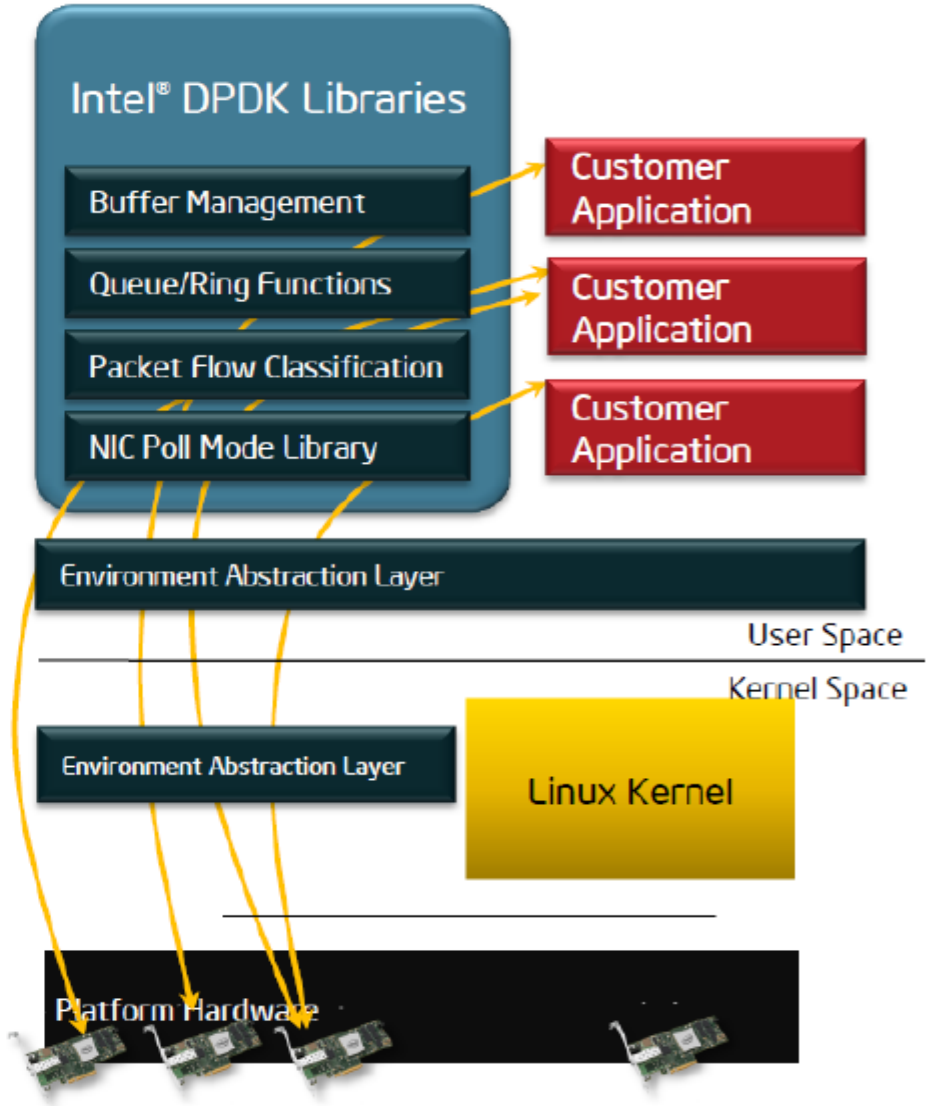


DPDK OVERVIEW

Runtime environment with low overhead

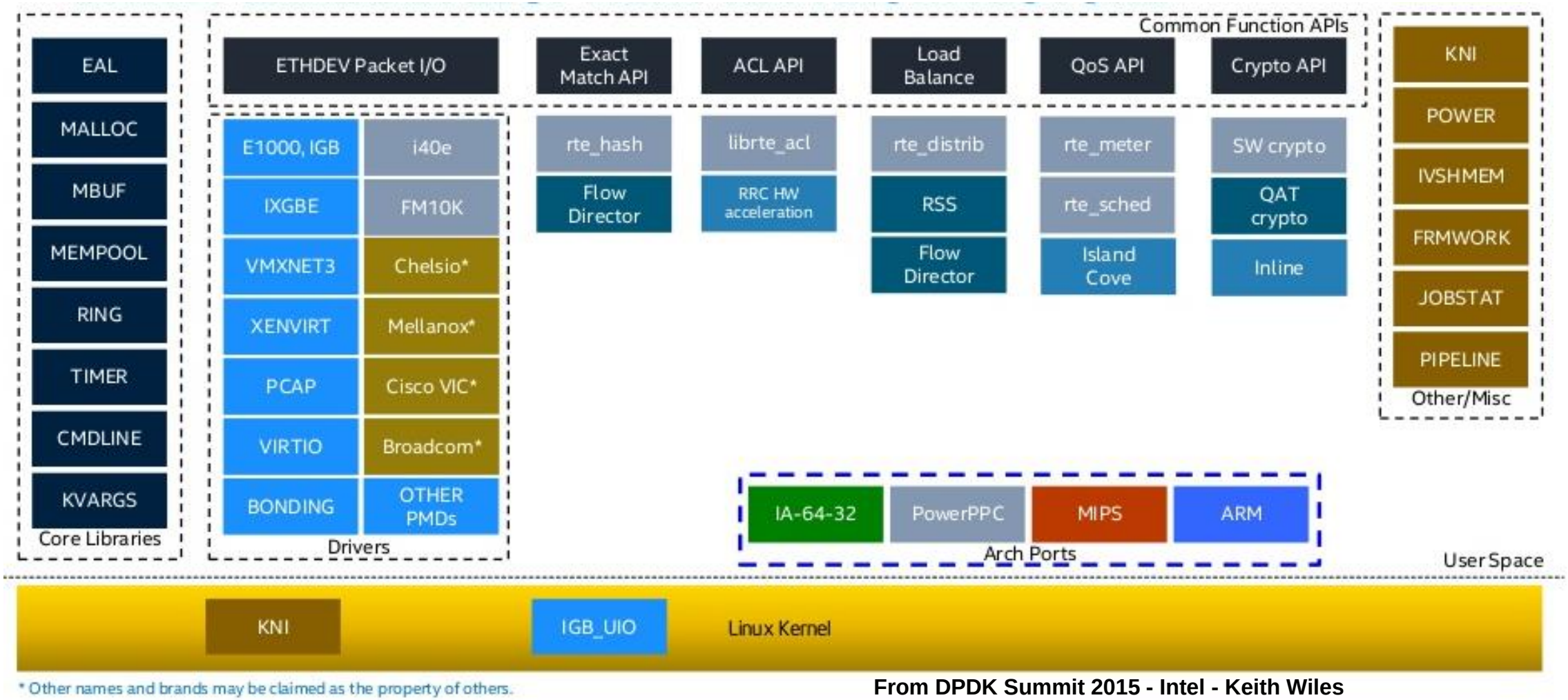
- Dataplane libraries run in userspace
  1. Memory management
  2. Buffer management
  3. Custom driver
  4. ...
- Environment Abstraction Layer (EAL)

”Easy to use.” - Intel



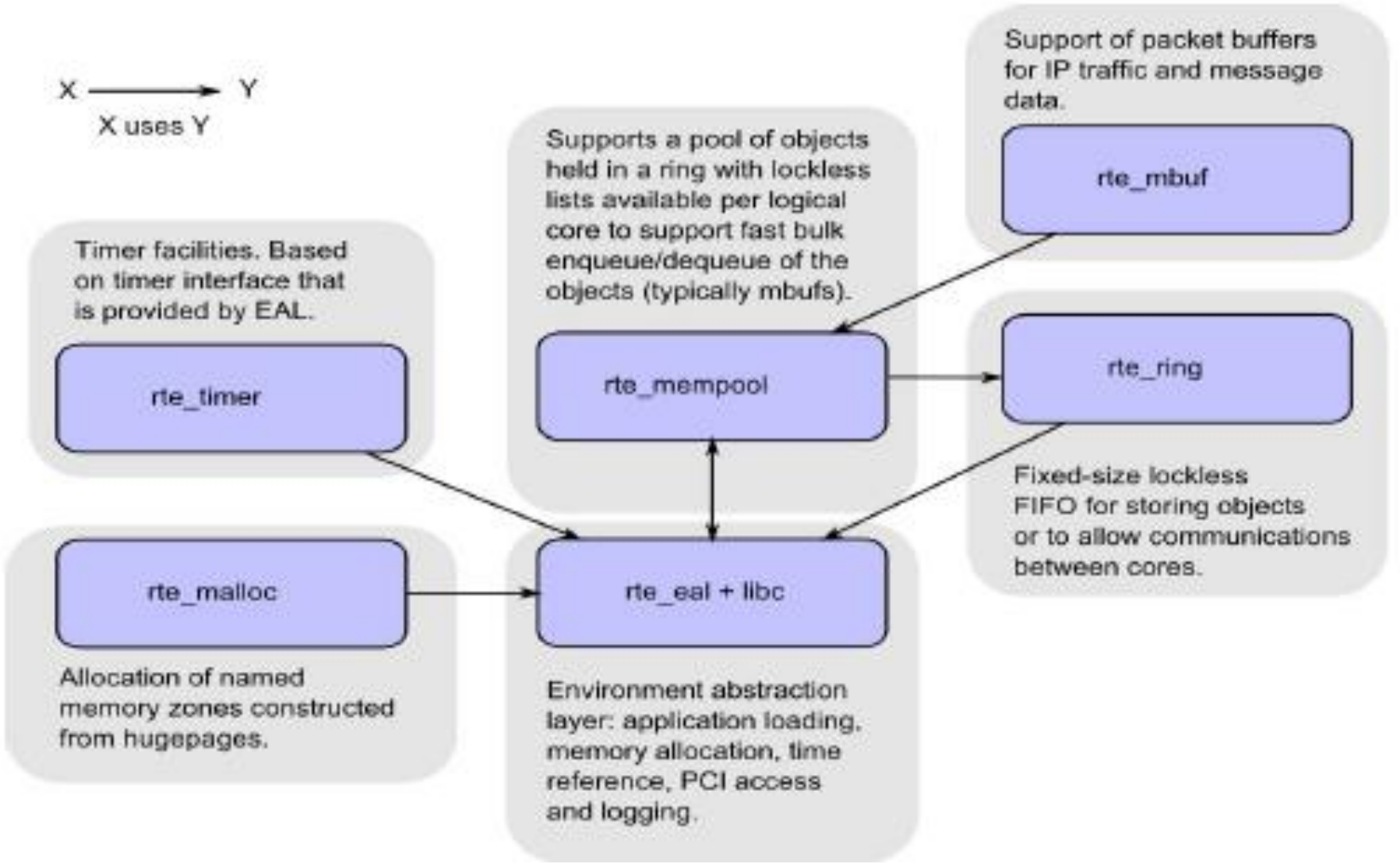
Source: “A Look at Intel’s Dataplane Development Kit” by Dominik Scholz

DPDK Architecture



From DPDK Summit 2015 - Intel - Keith Wiles
http://www.slideshare.net/jstleger/dpdk-summit-2015-intel-keith-wiles

Core Components Architecture

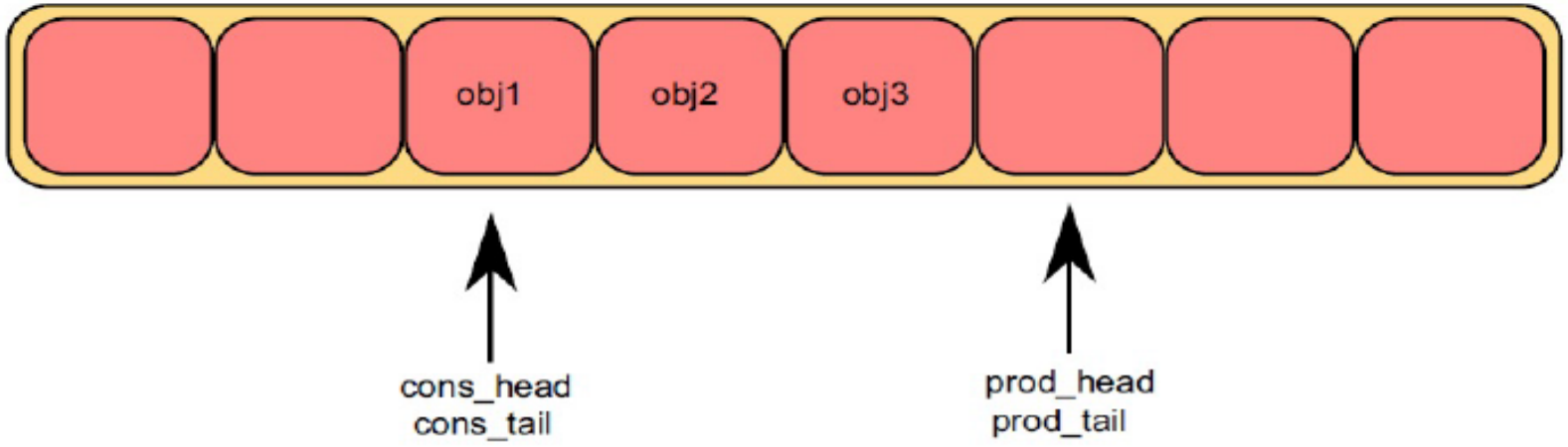


Queue Manager

Fixed-sized ring implemented as table of pointer to any object

Properties:

- FIFO
- Lockless (no active waiting)
- Supports multi consumer/producer enqueue/dequeue scenarios
- Supports bunch-processing of objects

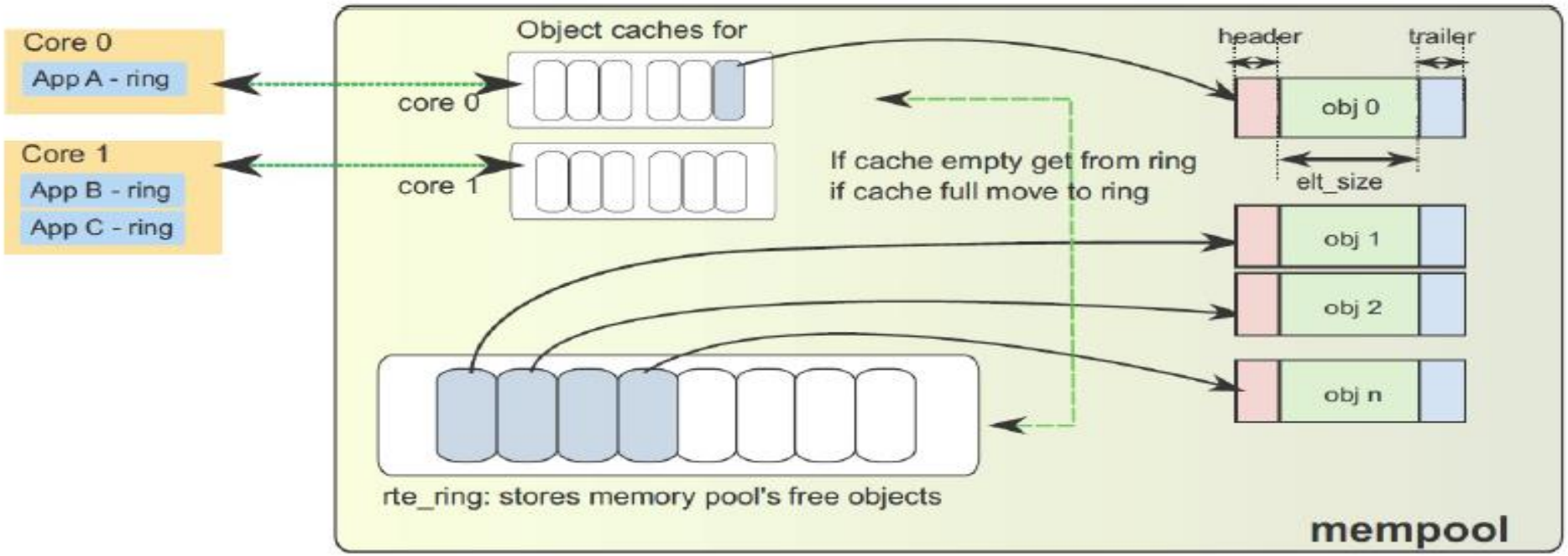


Source: "A Look at Intel's Dataplane Development Kit" by Dominik Scholz

Memory Manager

mempool structure:

- Pool of fixed-sized objects
- Uses a ring to store free objects
- Per core cache (optional)



Source: "A Look at Intel's Dataplane Development Kit" by Dominik Scholz

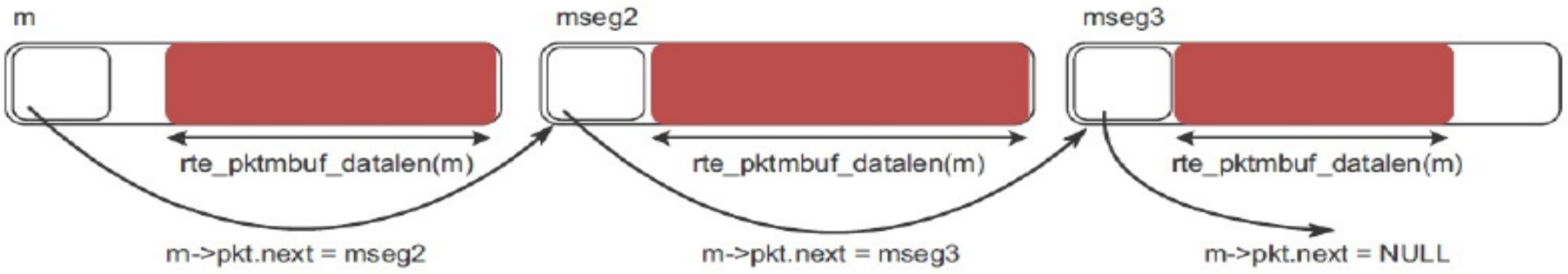
Buffer Manager

mbuf structure used to store network packets

- Created before runtime
- ”Allocation”: take a free mbuf from a mempool
- ”Deallocation”: put the mbuf back to the mempool
- Small size to fit in one cache-line ($\rightarrow$ mbuf-chaining)

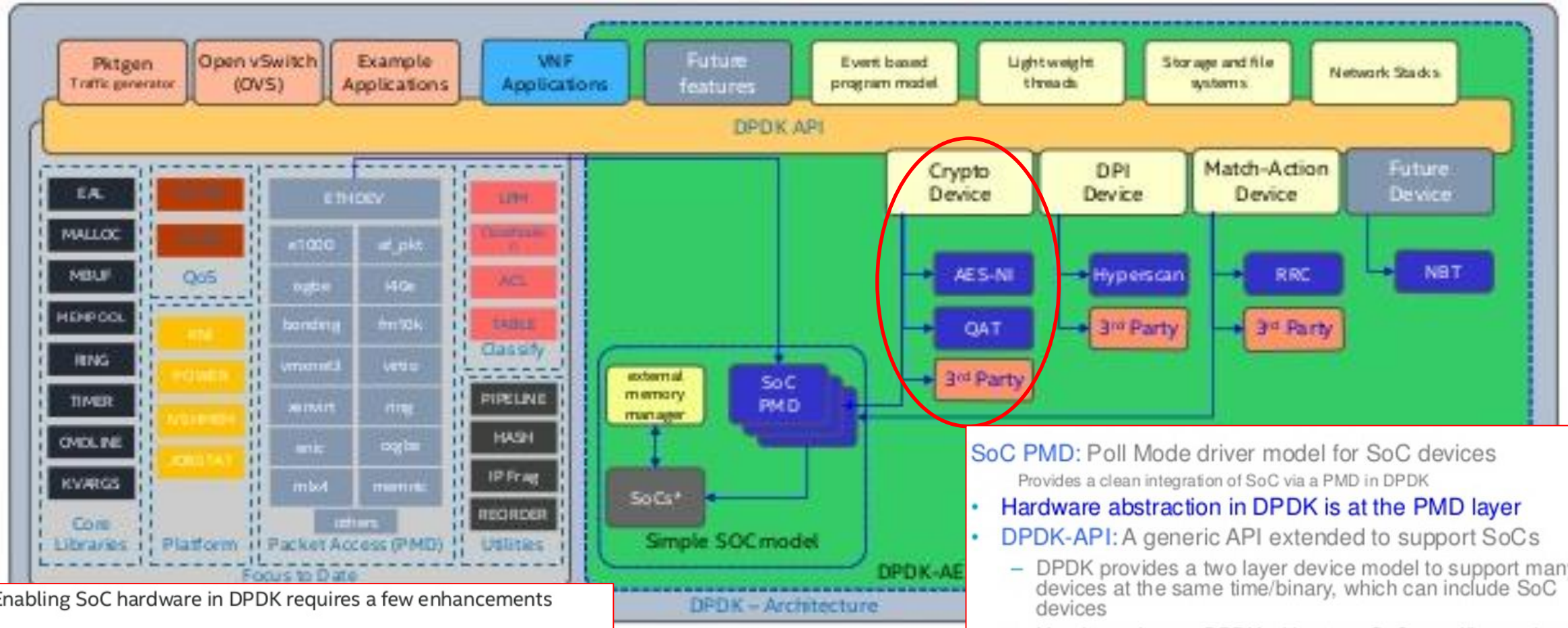
mbuf contains:

1. Metadata: control information, e.g. packet length
2. Pointer to next mbuf
3. Packet data: header and payload



Source: “A Look at Intel’s Dataplane Development Kit” by Dominik Scholz

Work In Progress: DPDK-AE (Acceleration Enhancements)



Enabling SoC hardware in DPDK requires a few enhancements

- Need a way to configure these non-PCIe devices
- Add support to DPDK mempool's to allow for external or hardware memory managers
- Add support for event based applications
  - e.g. Open Event Machine or others to utilize an event based programming model

From DPDK Summit 2015 - Intel -
http://www.slideshare.net/jstleger/dp

SoC PMD: Poll Mode driver model for SoC devices

Provides a clean integration of SoC via a PMD in DPDK

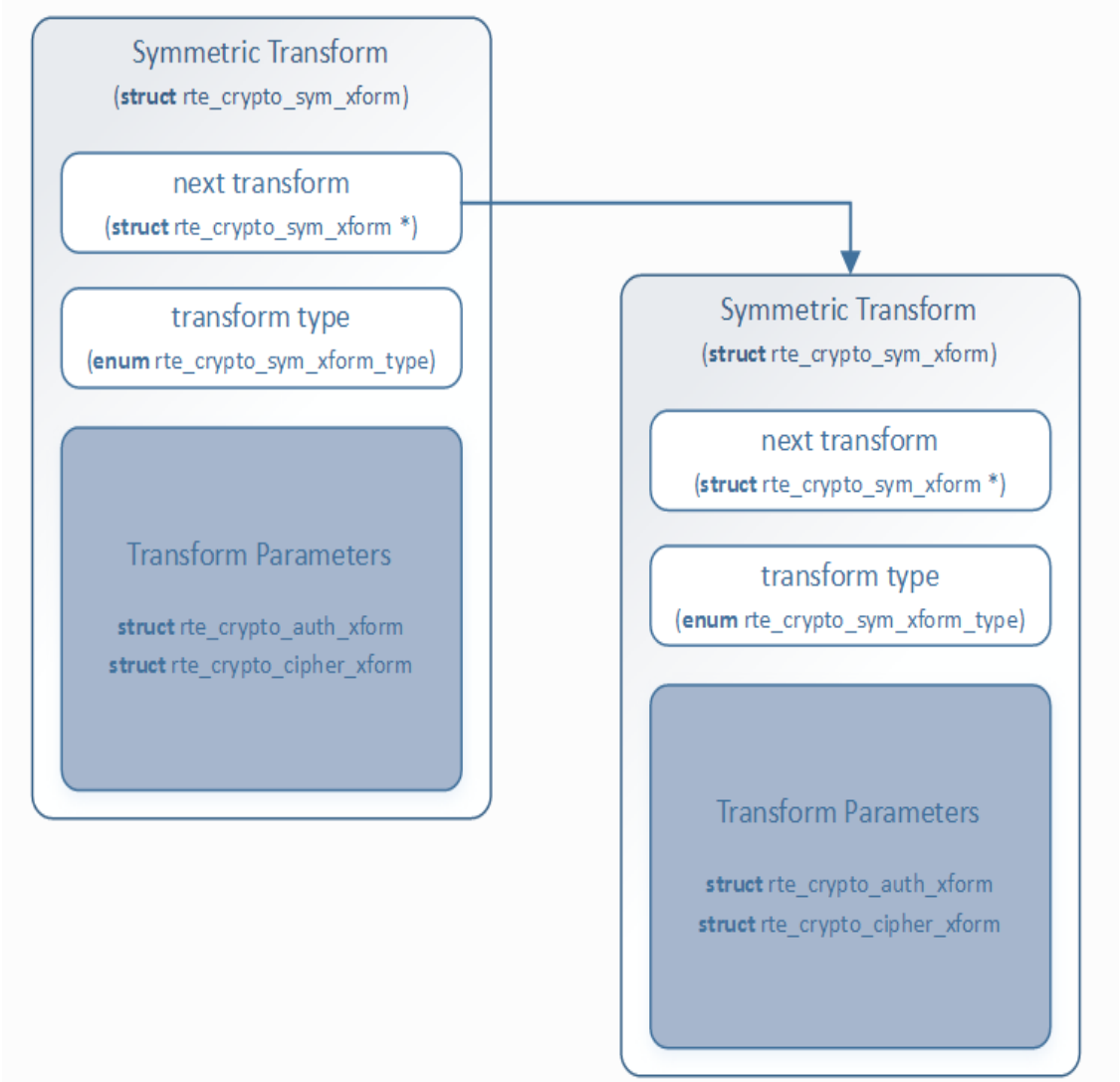
- Hardware abstraction in DPDK is at the PMD layer
- DPDK-API: A generic API extended to support SoCs
  - DPDK provides a two layer device model to support many devices at the same time/binary, which can include SoC devices
  - Need to enhance DPDK with some SoC specific needs or features to support SoC hardware
    - Non-PCI configuration
    - External memory manager(s) (for hardware based memory)
    - Event based programming model
- SoC-PMD: Poll Mode Driver model for SoC
  - Allows SoC SDK's to remain private

DPDK Crypto Subsystem

- Session-less Mode
  - For each job, software defines;
    - The data to be operated upon (input buffers, lengths, offsets)
    - The output buffers to hold results
    - The cryptographic operations to be performed
    - Keys & context for the cryptographic operations
- Session Oriented Mode
  - For each job, software defines;
    - The data to be operated upon (input buffers, lengths, offsets)
    - The output buffers to hold results
  - Cryptographic operations, keys & context are defined at session establishment time, and referenced for each job
- Supports virtual and physical crypto devices
  - Virtual Device (Software Implementation)
    - Intel AES-NI/vector operations
    - ARM NEON instructions *
  - Physical Device (Hardware Accelerated)
    - QAT
    - DPAA-CAAM*
    - DPAA2-CAAM*
- Test Applications
  - L2fwd with crypto
  - ipsec forward application

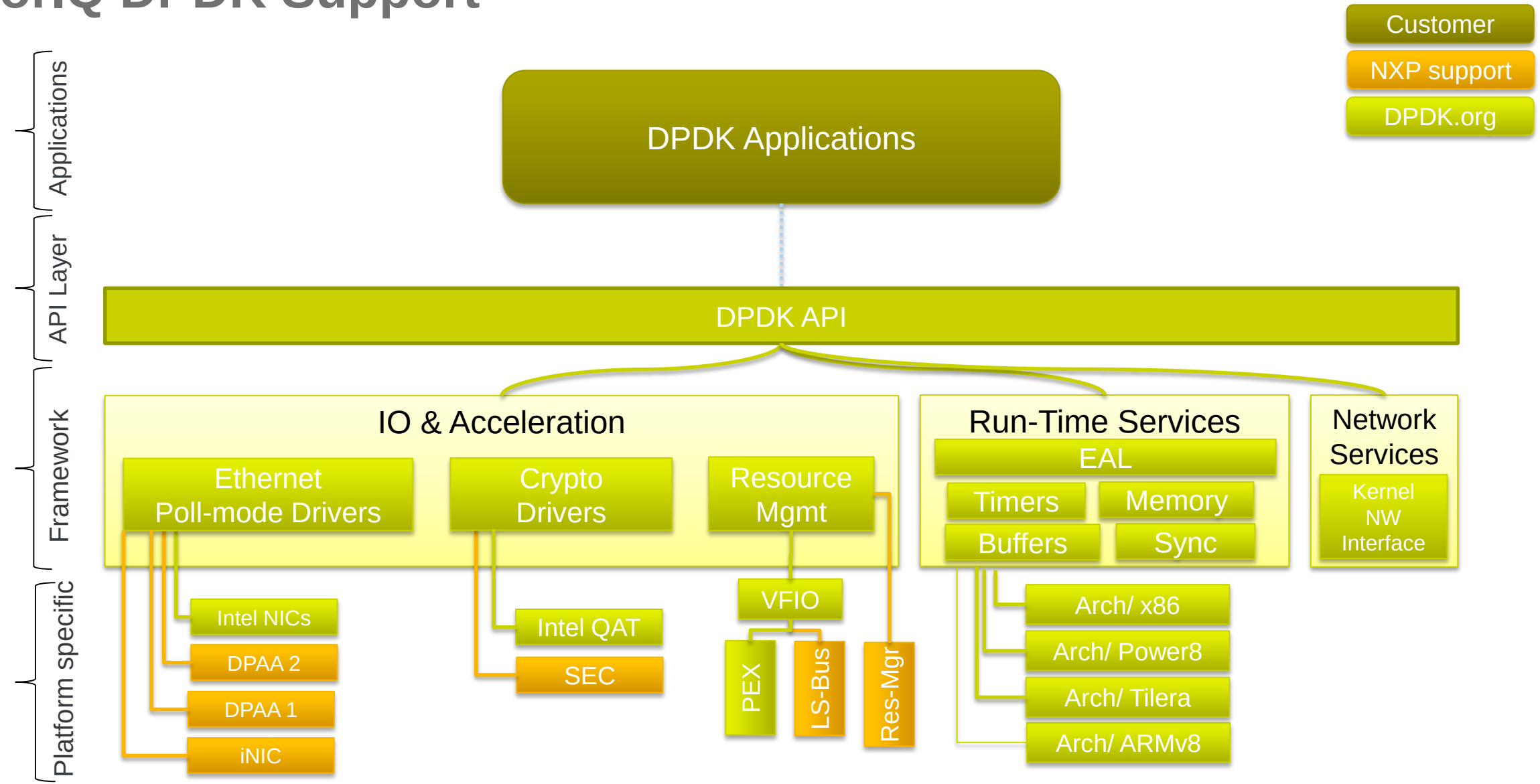
DPDK Crypto APIs

- device creation and configuration
  - rte_cryptodev_configure, rte_cryptodev_queue_pair_setup
- device capabilities.
  - rte_cryptodev_info_get
- Pool creations
  - rte_crypto_op_pool_create, rte_crypto_op_alloc
- Session Management
  - rte_cryptodev_sym_session_create
  - rte_cryptodev_sym_session_free
- Packet operations
  - rte_cryptodev_enqueue_burst
  - rte_cryptodev_dequeue_burst

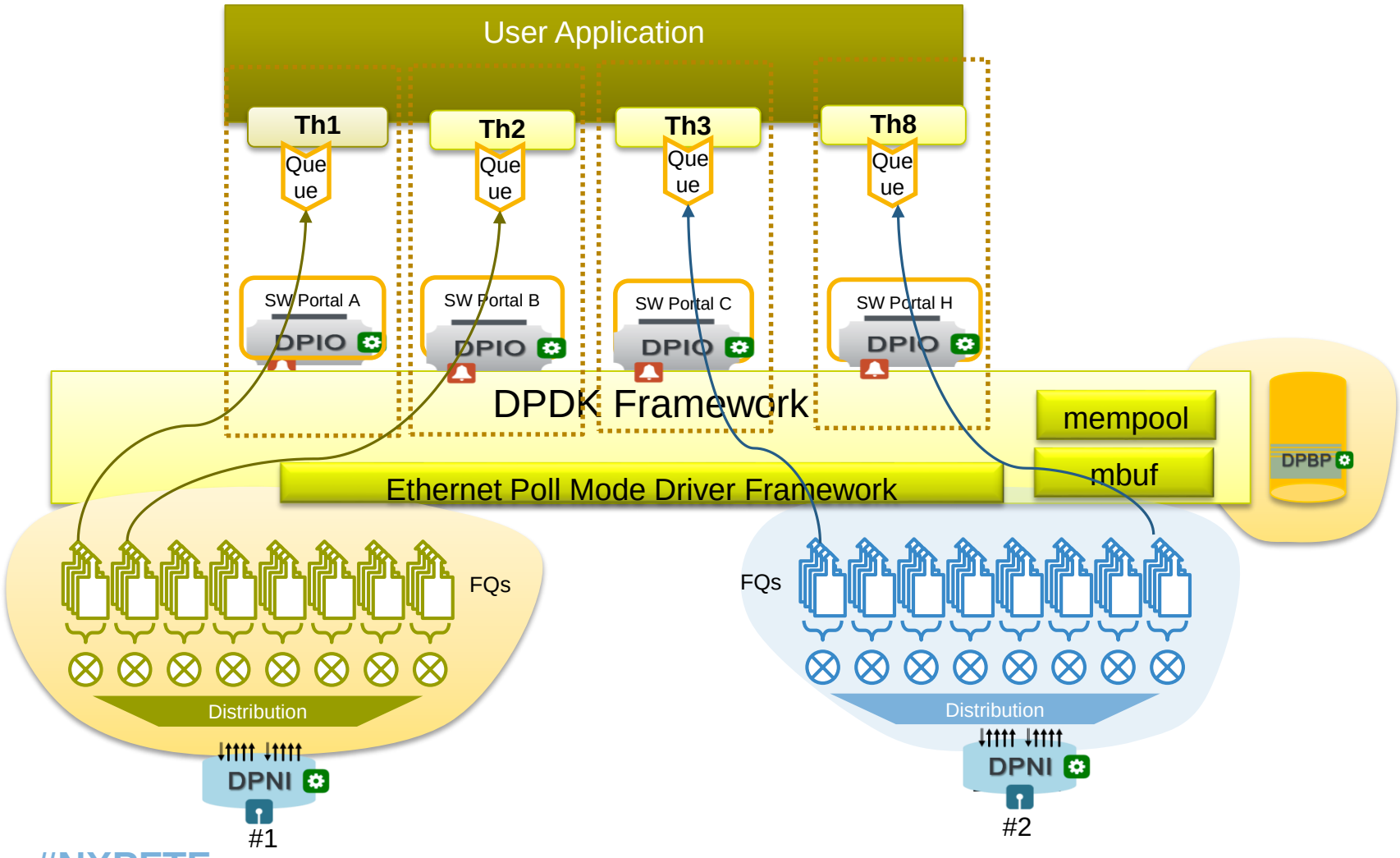


DPDK IMPLEMENTATION ON NXP PLATFORM

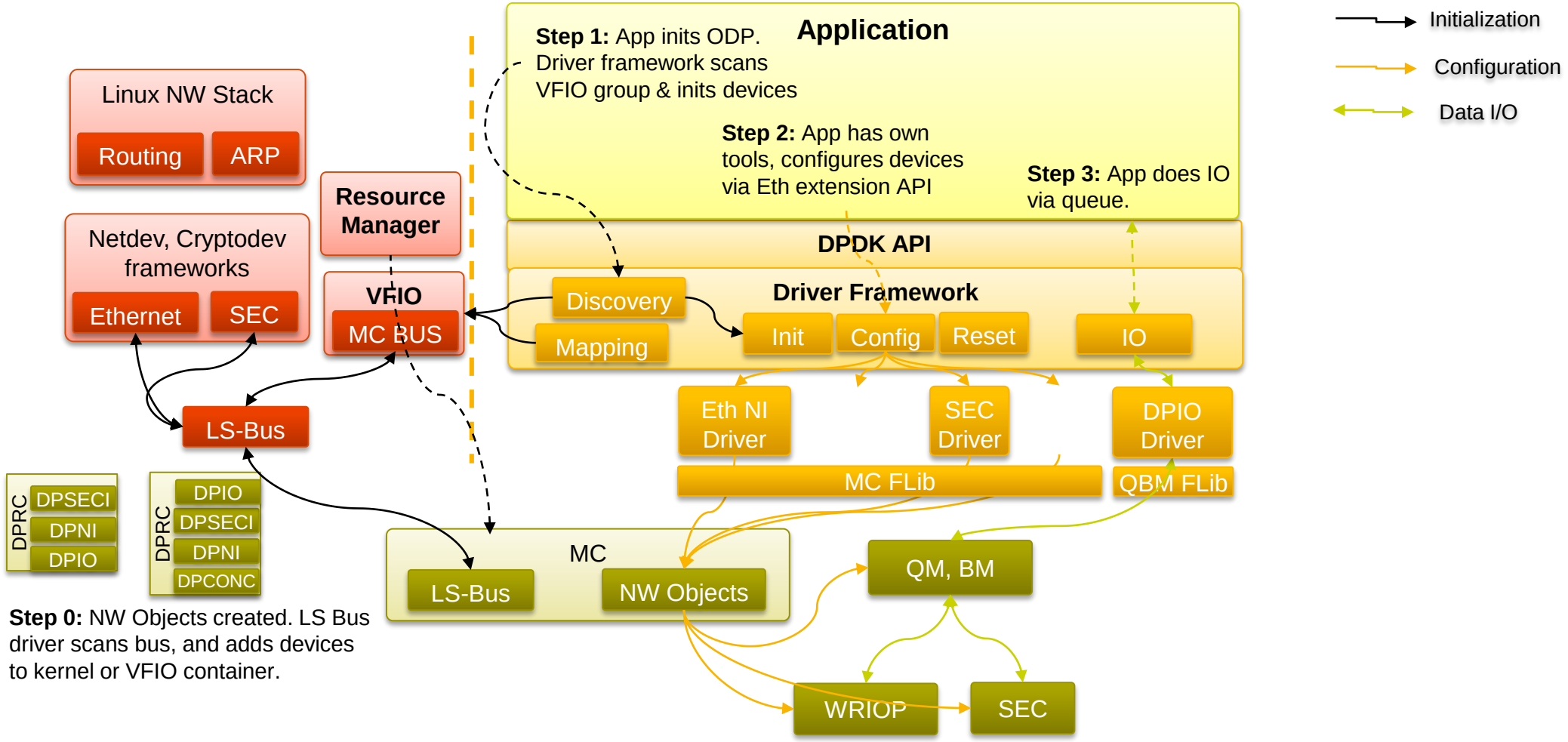
QorIQ DPDK Support



DPDK: Integration with DPAA2 framework



DPDK – Device Management with VFIO



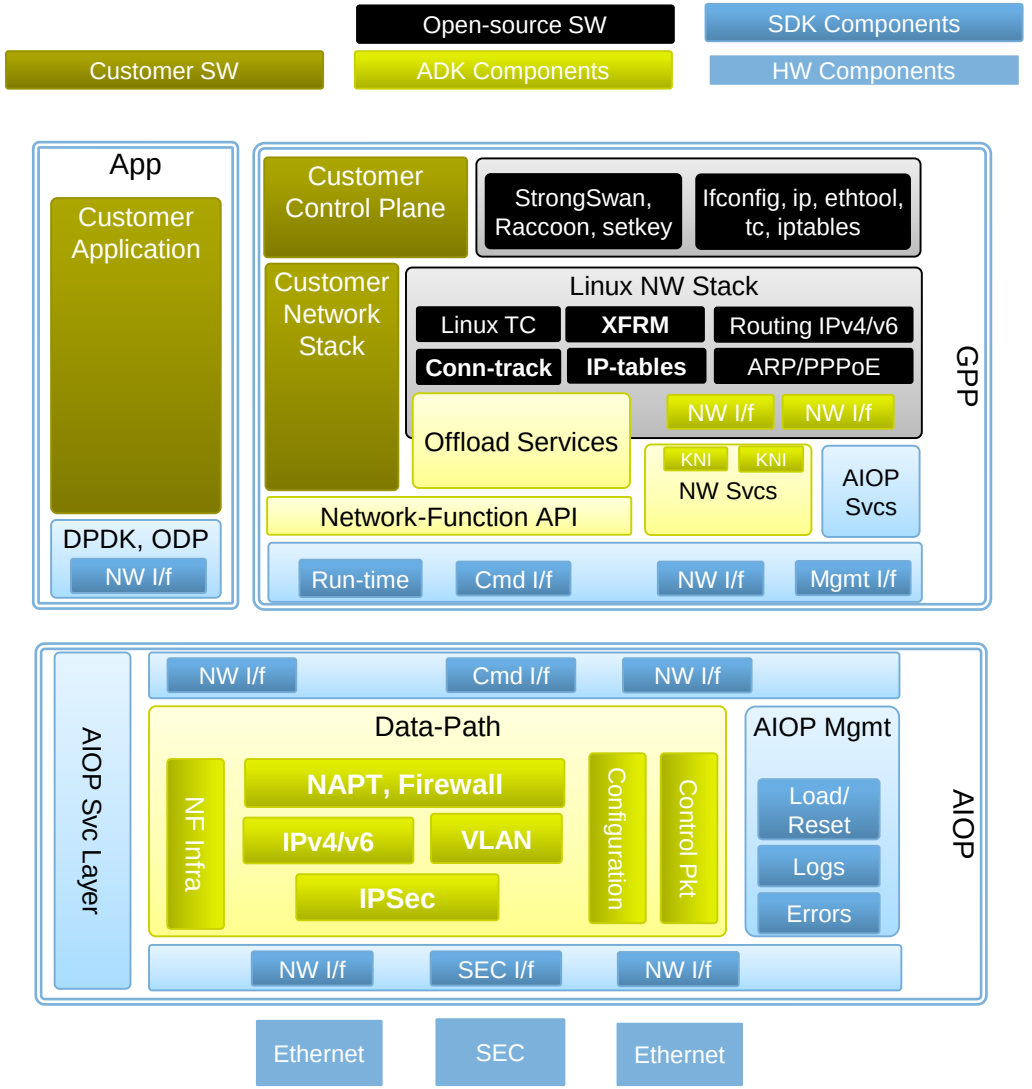
APPLICATION INTEGRATION

Networking & Security Package

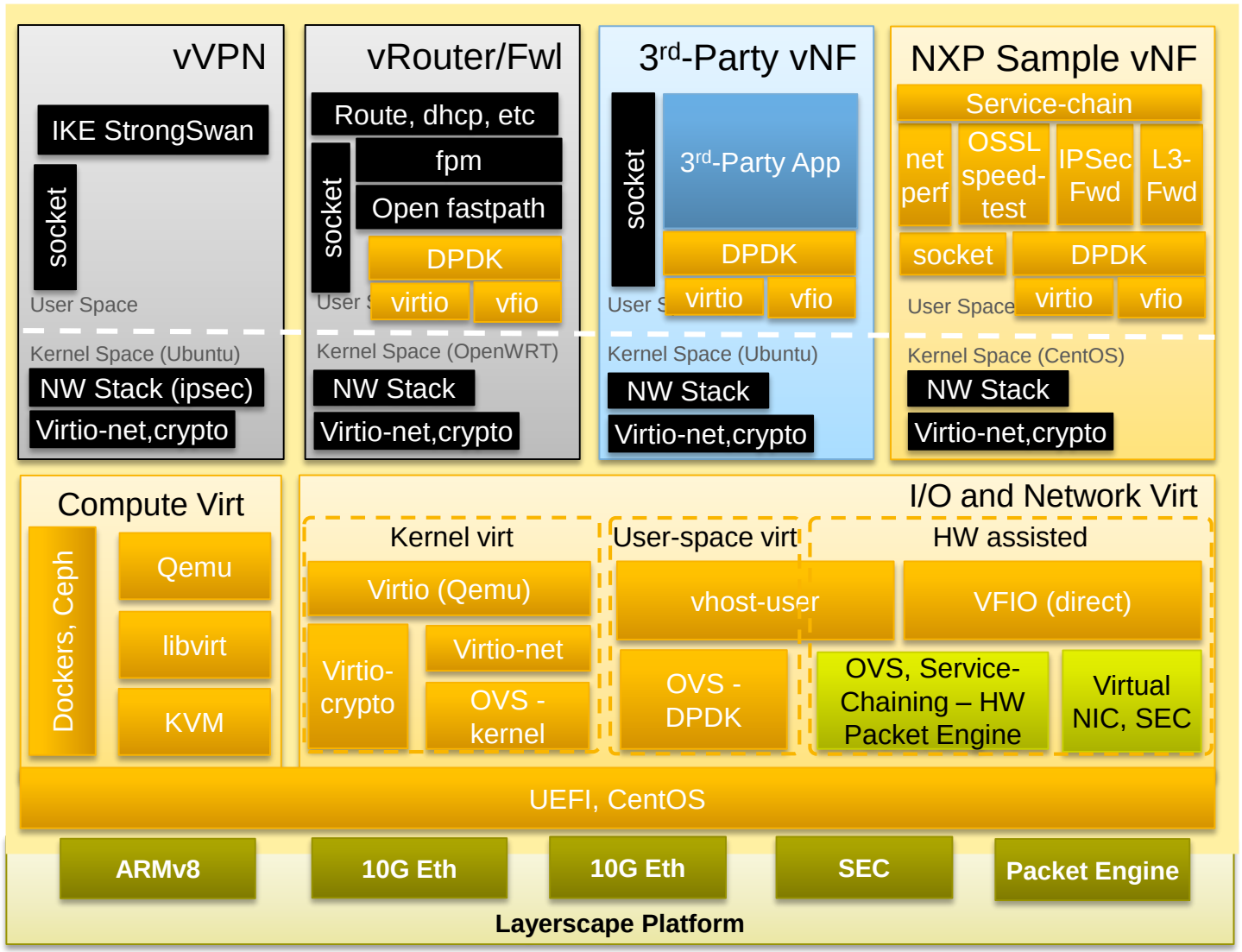
- Features
  - Data-path
    - IPv4/v6 Routing/Forwarding
    - NAPT, Firewall, IPSec
  - Control-Path
    - Network-Function API
    - Integrated with Linux NW stack
    - Seamless integration with open-source SW like Strong-Swan, iptables etc.

- Intended deployment
  - SMB/Enterprise/WLAN Gateways
  - Services cards
  - Cell-site routers, Access concentrators.
  - Network functions – vCPE, vAccess

- NXP value proposition
  - Data-path entirely offloaded to HW, GPP free for customer apps
  - Highest performance/watt – maximum leverage of NXP HW offloads
  - Rich feature-set = fast deployment / time-to-market
  - Easy-to-use NF API to integrate with customer stacks



NFV Solution Architecture



OP-NFV (Orchestration, Mgmt)

- Fuel, Apex Installer
- Open Contrail, Daylight
- Open Stack

NFV Development Kit	
OP-NFV	• Brahmaputra
DPDK	• v2.2+
OVS	• v2.4/2.5 • OVS DPDK • OVS Packet-Engine
KVM	• v2.2
Qemu	• v2.5
Libvirt	• 1.2.20
Linux	• LTS Kernel 4.1.2
Orchestration	• Open Daylight
Reference vNFs	• Open Source • vRouter, • vFW (iptables), • vVPN (strongSwan)
Distro	• UEFI • CentOS

NXP enablement for NFV – upstreamed to community, competitive performance

Re-use from OP-NFV community and run un-modified

Re-use from 3rd-Party sources and run un-modified

NXP HW assists for extra performance

Session Summary

- SDN and NFV are market drivers for cross-platform API standards for efficient user space packet processing with hardware acceleration.
- ODP, originally from the ARM ecosystem, has rapidly emerged as cross-platform API for SoCs, allowing leverage of a range of acceleration features, including crypto.
- DPDK, from the IA ecosystem, started as IA-centric APIs for narrower range of accelerators, but with DPDK-AE, is beginning to support hardware accelerators.
- ODP & DPDK is rapidly emerging as the only cross-platform API standard with a fully engaged ecosystem.
- NXP is actively engaged in and is driving definition of ODP API within Linaro.
- NXP is committed to provide efficient implementations of the ODP API across QorIQ platforms.
- NXP also provide efficient implementation of DPDK across QorIQ platforms.

Resources

- Software support for LS208x - NXP SDK
- Open Data Plane (www.opendataplane.org)
- DPDK (www.dpdk.org)

Visit us in the Tech Lab – #245
Leveraging DPDK for Virtualized Network Functions on DPAA2.0

Visit us in the Tech Lab – #246
Accelerate User-space Networking with QorIQ Processors

Software Products and Services

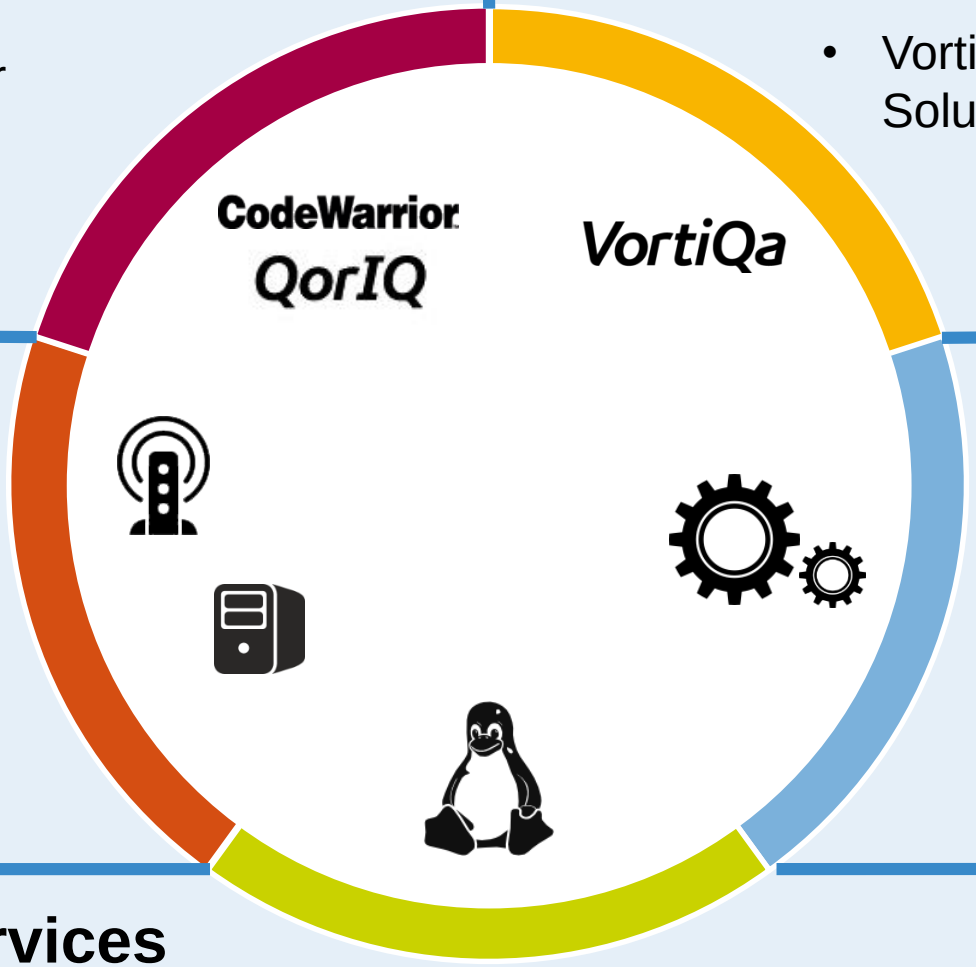
Visit us in the Tech Lab – #247

Development Tools

- CodeWarrior

Runtime Products

- VortiQa Software Solutions



Solutions Reference

- IOT Gateway
- OpenWRT+

Integration Services

- Security Consulting
- Hardened Linux

Linux® Services

- Commercial Support
- Performance Tuning

Accelerate **Customer Time-to-Market**

Deliver **Commercial Software, Support, Services and Solutions**

Simplify **Software Engagement with NXP**

Create **Success!**

Find us online at www.nxp.com/networking-services

SECURE CONNECTIONS
FOR A SMARTER WORLD

ATTRIBUTION STATEMENT